S25 CSCI 4210 — Operating Systems — Exam 1
SOLUTIONS — (document version 1.0) ♣ ♢ ♡

Overview

- This is a 110-minute exam (6:00-7:50PM); if you have extra-time accommodations, then you have either 165 minutes (50%) or 220 minutes (100%) and should write your start and end times at the top of this page for reference by you and the exam proctors
- Questions will **not** be answered except when there is a glaring mistake or ambiguity in the statement of a question; we **cannot** clarify a question for you; please do your best to interpret and answer each question
- There are **nine** multiple choice questions, **three** short answer questions, and **one** extra-credit question on this exam; this exam will be graded out of 50 points, with a 51 possible
- For multiple choice questions, no partial credit will be given; for short answer questions, be clear and concise in your answers
- No calculators, cellphones, etc.; please be sure to turn off electronic devices before the exam
- When done, only submit your double-sided exam answer sheet; **proofread your work before handing in your paper**; anything written on the stapled exam will **not** be graded
- You are allowed to use one double-sided or two single-sided 8.5"×11" cribsheets
- Use the last three pages of the stapled exam for tinkering, scrap work, etc.
- **All work on this exam must be your own; do not copy or communicate with anyone else about this exam until the exam is graded**
- Once we have graded all exams, solutions will be posted; the grade inquiry window (in Submitty) for this exam will then be one week

Assumptions

1. Assume that all given code compiles without warnings or errors (unless otherwise noted)
2. Assume that all library function and system calls return successfully (unless otherwise noted)
3. Assume that all code runs on a 64-bit Ubuntu 22.04.5 LTS instance
4. Assume that processes initially run with a process ID (`pid`) of 777; child process IDs are then numbered sequentially as they are created starting at 800 (i.e., 800, 801, 802, etc.)

1. **(3 POINTS)** In the code below, assume the `data.txt` file exists and is readable; also assume the `q1.txt` file does not initially exist. When you run this code, which file descriptor is associated with the write end of the pipe? Select the **best** answer.

```
int main()
{
  int x = 1;
  int * q1 = malloc( 2 * sizeof( int ) );
  close( 0 );
  close( x );
  int y = open( "data.txt", O_RDONLY );
  pipe( q1 );
  read( y, &x, sizeof( int ) );
  open( "q1.txt", O_WRONLY | O_CREAT, 0600 );

  /* which descriptor is the write end of the pipe? */

  return EXIT_SUCCESS;
}
```

SOLUTION: 3

2. **(3 POINTS)** How many total bytes are allocated on the runtime heap when we reach the one `printf()` call in the code shown below? Select the **best** answer.

```
int main()
{
  short ** q2 = calloc( 2, sizeof( short * ) );
  *(q2 + 1) = calloc( 8, sizeof( short ) );
  q2 = realloc( q2, 8 * sizeof( short * ) );
  *(q2 + 3) = malloc( 17 * sizeof( short ) );

  /* how many bytes are allocated on the heap at this point? */
  printf( /* ... */ );

  free( q2 );
  return EXIT_FAILURE;
}
```

SOLUTION: 114 bytes

3. **(3 POINTS)** In the code below, assume the `data.txt` and `q3.txt` files both exist and are both readable and writable. When you run this code, how many bytes are written to the `q3.txt` file? Select the **best** answer.

```
int main()
{
  int q3 = open( "q3.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );
  dup2( q3, 1 );
  open( "data.txt", O_RDONLY );
  close( 0 );
  write( q3, "THREE-PEAT!!!", 3 );
  printf( "Q3-%d\n", getpid() );
  fprintf( stderr, "Q3--\n" );
  close( q3 );
  return EXIT_SUCCESS;
}
```

SOLUTION: `10`

4. **(3 POINTS)** Given the code below, how many times is the substring `"fork"` printed to the terminal? Select the **best** answer.

```
int main()
{
  char * q4 = malloc( 4 );
  sprintf( q4, "fork" );
  fork();
  printf( "%s?", q4 );
  if ( fork() == 0 ) fork();
  printf( "%s!\n", q4 );
  free( q4 );
  return EXIT_SUCCESS;
}
```

SOLUTION: `12`

5. **(3 POINTS)** How many total bytes are allocated on the runtime stack when we run the code shown below? Select the **best** answer.

```
int main()
{
  int ** q5 = calloc( 5, sizeof( int * ) );
  *q5 = calloc( 5, sizeof( int ) );
  int x = 2, y = 4;
  *((*q5) + y) = y;
  x += **q5;
  int * s = *(q5 + 1);
  s = calloc( x * y, sizeof( int ) );
  printf( "%d-%d\n", *((*q5) + y), *s );

  /* how many bytes are allocated on the runtime stack? */

  free( *q5 );
  free( q5 );

  return EXIT_SUCCESS;
}
```

SOLUTION: 24

Questions 6 and 7 use the code shown below.

```
void f6( int sig )
{
  printf( "PARADISE" );
  fflush( stdout );
}

int main()
{
  signal( SIGINT, f6 );
  close( 0 );
  close( 1 );
  int * q6 = calloc( 6, sizeof( int ) );
  void * A = q6 + 2;
  pipe( q6 );
  printf( "GO-S" );
  fflush( stdout );
  sleep( 6 );
  read( *q6, A, 7 );
  char * L = A;
  fprintf( stderr, "%sKS\n", L );
  close( *(q6) );
  close( *(q6 + 1) );
  free( q6 );
  return EXIT_SUCCESS;
}
```

6. **(3 POINTS)** What is the **exact** terminal output of the given code if the user hits `CTRL-C` during the `sleep()` call? Select the **best** answer.

   SOLUTION: `GO-SPARKS`

7. **(3 POINTS)** From Question 6 above, if the user hits `CTRL-C` during the `sleep()` call, how many bytes remain unread on the pipe when this code reaches the `return` statement? Select the **best** answer.

   SOLUTION: `5` (`"ADISE"`);

8. **(3 POINTS)** If the given code below is executed as `./a.out > q8.txt`, how many possible outputs are there for the resulting `q8.txt` file? Select the **best** answer.

```
int main()
{
  int q8 = 8;
  pid_t p = fork();
  if ( p == 0 )
  {
    q8 *= 8;
    printf( "CHILD: q8 is %d\n", q8 );
  }
  else if ( p > 0 )
  {
    q8 += 8;
    printf( "PARENT: q8 is %d\n", q8 );
    waitpid( p, &q8, 0 );
    printf( "PARENT: exit status is %d\n", WEXITSTATUS( q8 ) );
  }
  return q8;
}
```

SOLUTION: `1`

9. **(3 POINTS)** In the code below, assume the `q9.txt` file does not initially exist. What are the permissions applied to the creation of the `q9.txt` file, as would be shown by an `ls -l` shell command? (Ignore `umask` for this question.) Select the **best** answer.

```
int main()
{
  int q9 = open( "q9.txt", O_WRONLY | O_CREAT, 0644 );
  write( q9, "Q9Q9Q9Q9Q9", 9 );
  close( q9 );
  return EXIT_SUCCESS;
}
```

SOLUTION: `-rw-r--r--`

10. **(12 POINTS)** Consider the given code below, which includes line numbers for part (c). Answer the questions below on your answer sheet. Note that at least one of the processes blocks forever when the given code runs. Also note that the `/usr/bin/grep` program reads from /verb+stdin+ and displays on `stdout` each line that contains the specified substring.

```
 1.   int main()
 2.   {
 3.     int * q10 = malloc( 2 * sizeof( int ) );
 4.     close( 0 );
 5.     int rc = pipe( q10 );
 6.     rc = fork();
 7.     int x = *q10, y = *(q10 + 1);
 8.     free( q10 );
 9.     printf( "%u+%d+%d\n", getpid(), x, y );
10.     if ( rc > 0 )
11.     {
12.       close( x );
13.       rc = write( y, "abc\ndefg\nhijk\nlmn\nopgr\nstuv\nwxgz\n", 18 );
14.       printf( "A%d-", rc );
15.       rc = waitpid( -1, NULL, 0 );
16.       printf( "B%d\n", rc );
17.       return EXIT_SUCCESS;
18.     }
19.     execl( "/usr/bin/grep", "grep", "g", NULL );
20.     perror( "execl() failed" );
21.     return EXIT_FAILURE;
22.   }
```

(a) What is the **exact** terminal output of the given code? If multiple outputs are possible, succinctly describe all possibilities. If a process blocks forever, indicate this with `<parent-blocks>` and/or `<child-blocks>`.

SOLUTION: (9pts) There are three lines of output, graded as follows:

```
                     -fork()-
                    /        \
                   /          \
----------<parent>--      --<child>-----------
777+0+3               <==> 800+0+3
                      <==> defg
--------------------      --------------------
<parent-blocks>           <child-blocks>


[2pts]    777+0+3  <== award 0.5pts for each value
[2pts]    800+0+3  <== award 0.5pts for each value
[1pt]     defg
[2pts]    interleaving occurs as shown above
[1pt]     <parent-blocks>  <== due to the waitpid() call
[1pt]     <child-blocks>   <== due to the two open write descriptors
                               on the pipe (in the parent process)

[subtract -0.5pts for each out-of-order or incorrect line or error]
```

(b) If the `execl()` called fails and `grep` is not executed, which process, if any, blocks forever? Answer this by writing `<parent-blocks>` or `<child-blocks>` or `<neither-block>`.

SOLUTION: (1pt) `<neither-block>`

(c) Add code to ensure that both processes terminate successfully without any possibility of blocking forever. To indicate where to insert your code, specify line numbers with a comment, as shown in the example below. Always reference the original line numbers shown above.

```
/* add between lines 5 and 6: */
int xyz = 123;
```

Do **not** change or remove any of the given code. Do **not** use square brackets.
And do **not** use `return`, `exit()`, `_exit()`, `abort()`, or any other means to force an exit.

SOLUTION: (2pts) The parent process must close the write end of the pipe, or else the parent process blocks forever on the `waitpid()` call.

No partial credit if missing line numbers where the `close()` calls should be added. Only award 0.5 points if file descriptor is hard-coded, i.e., `close( 3 )`.

```
[1pt]    /* must be between lines 13 and 15: */
         close( y );    /* close the write end of the pipe */

[1pt]    /* must be between lines 18 and 19: */
         close( y );    /* close the write end of the pipe */
```

(If other answers are given, "test" the code by tracing through it to see if the solution has any merit.)

11. **(6 POINTS)** Apply the Shortest Remaining Time (SRT) scheduling algorithm to the processes described in the table below. For each process, calculate the turnaround time, the wait time, and the number of times each process is preempted.

If necessary, break "ties" based on process ID order (i.e., lowest to highest).

Ignore context switch times and other such overhead.

Fill in the table on your answer sheet with your answers.

The grid further below is optional for you to use.

SOLUTION: Award **0.5 points** per cell.

	CPU Burst	**Arrival**	**Turnaround Time**	**Wait Time**	**Preemptions**
P1	8 ms	0 ms	11 ms	3 ms	1
P2	3 ms	1 ms	3 ms	0 ms	0
P3	11 ms	4 ms	26 ms	15 ms	0
P4	8 ms	5 ms	14 ms	6 ms	0

```
        p
     +--+----+-------+--------+-----------+
SRT: |P1| P2 |  P1   |   P4   |    P3     |
     +--+----+-------+--------+-----------+
     0  1    4       11       19          30
```

12. **(5 POINTS)** Memory leaks and buffer overflows exist in some of the code on this exam.

   (a) How many bytes are dynamically allocated and not deallocated via `free()` throughout this exam? Only include memory **directly allocated** by the given code, i.e., ignore memory that is dynamically allocated by `printf()`, etc. Also ignore any code that you add for Question 10(c).

   Be sure to count **all** memory leaks in **all** running processes.

   Write the total number of bytes as a number (e.g., `1234`) on your answer sheet.

   SOLUTION: (3pts) 90 bytes

   If the correct answer is not given, then partial credit possibly might be awarded; if individual questions are referred to and correct, award 1 point per correct answer, as shown below.

```
[1pt]   Q1 --- memory leak: 8 bytes
[1pt]   Q2 --- memory leak: 50 bytes
        Q3 --- no memory leaks or buffer overflows
        Q4 --- no memory leaks --- buffer overflow
[1pt]   Q5 --- memory leak: 32 bytes
        Q6/Q7 --- no memory leaks or buffer overflows
        Q8 --- no memory leaks or buffer overflows
        Q9 --- no memory leaks or buffer overflows
        Q10 --- no memory leaks or buffer overflows

[-1pt for indicating a memory leak on another question]
```

   (b) Exactly one question on this exam has code that runs successfully but contains a buffer overflow. Which question has the buffer overflow? In one succint sentence, describe what causes the buffer overflow.

   SOLUTION: (2pts)

```
[1pt]  Q4 --- BUFFER OVERFLOW !!!!!!!
[1pt]  sprintf() writes 5 bytes to a 4-byte chunk of memory
```

13. **(1 POINT EXTRA CREDIT)** For extra credit, write **exactly** one line of code (i.e., use only one semicolon `';'` character) in `main()` that will always cause a segmentation fault. Your code must compile without warnings or errors.

```
int main() {                                        }
           /* ^^^ your one line of code goes here */
```

SOLUTION: (+1pt) Must be only one semicolon. Some possible answers are shown below. Check to be sure no compiler errors or warnings occur using `-Wall`. Be careful of `fork()`-bombs!

```
main();  /* death by recursion */
--or--
return *((int *)NULL); /* dereference a NULL pointer (any type except void) */
--or--
return *((int *)0); /* dereference a NULL pointer (any type except void) */
--or--
printf( "%d", *((int *)NULL) );   /* or something similar... */
```