E1 S18

Name: [redacted] RCS ID: [redacted] @rpi.edu

CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Spring 2018 Exam 1 (February 22, 2018)

Problem	Maximum Points	Your Score
1	3	3
2	3	0
3	10	10
4	23	15
5	23	11
6	23	6
7	15	11
8	+1 extra credit	0
	Total (100 points):	56

- You have 105 minutes to complete this exam (i.e., 10:00-11:45AM); note that 50% extra time is 160 minutes and 100% extra time is 210 minutes.
- Exam 1 will count as 10% of your final course grade.
- 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, electronic devices, etc.
- There are no syntax errors anywhere on this exam. Note that `#include` directives are omitted to save space on the page.
- Questions will not be answered except when there is a glaring mistake or ambiguity in the statement of a question.
- There are **seven questions** on this exam, plus one extra credit question. Please do your best to answer all questions.
- For C programming questions, assume that `fork()`, `wait()`, `waitpid()`, `read()`, `write()`, `getpid()`, `open()`, `close()`, `printf()`, `fprintf()`, and `pipe()` all return successfully.
- Also assume that the initial (parent) process ID is 777, with child processes, if any, numbered sequentially 888, 889, 890, etc.
- Exam 1 will be handed back and reviewed on either Thursday, March 1, 2018 or Monday, March 5, 2018.
- Grade grievances must be made no later than Thursday, March 8, 2018 by handing your exam back to Professor Goldschmidt (after class or during office hours).

1. **(3 POINTS)** In which CPU scheduling algorithms may starvation occur? Circle the best answer (i.e., only circle one answer).

(a) RR
(b) FCFS
(c) SJF
(d) Both (a) and (b), but not (c)
(e) Both (b) and (c), but not (a)
(f) Both (a) and (c), but not (b)

2. **(3 POINTS)** If the file descriptor table has active descriptors 1, 2, and 3 for a given process P, which two descriptors will the `pipe(p)` system call assign to input argument p? Circle the best answer (i.e., only circle one answer).

(a) `p[0]==1` and `p[1]==3`
(b) `p[0]==4` and `p[1]==0`
(c) `p[0]==0` and `p[1]==4`
(d) `p[0]==3` and `p[1]==1`
(e) `p[0]==3` and `p[1]==4`
(f) None of the above

3. **(10 POINTS)** For the Shortest Remaining Time (SRT) scheduling algorithm, given actual CPU burst times for process P shown below and an initial guess of 8 ms, calculate the estimated next CPU burst times for P by using exponential averaging with an α of 0.5.

$T(n) = \alpha * T(n-1) + \alpha * t(n-1)$

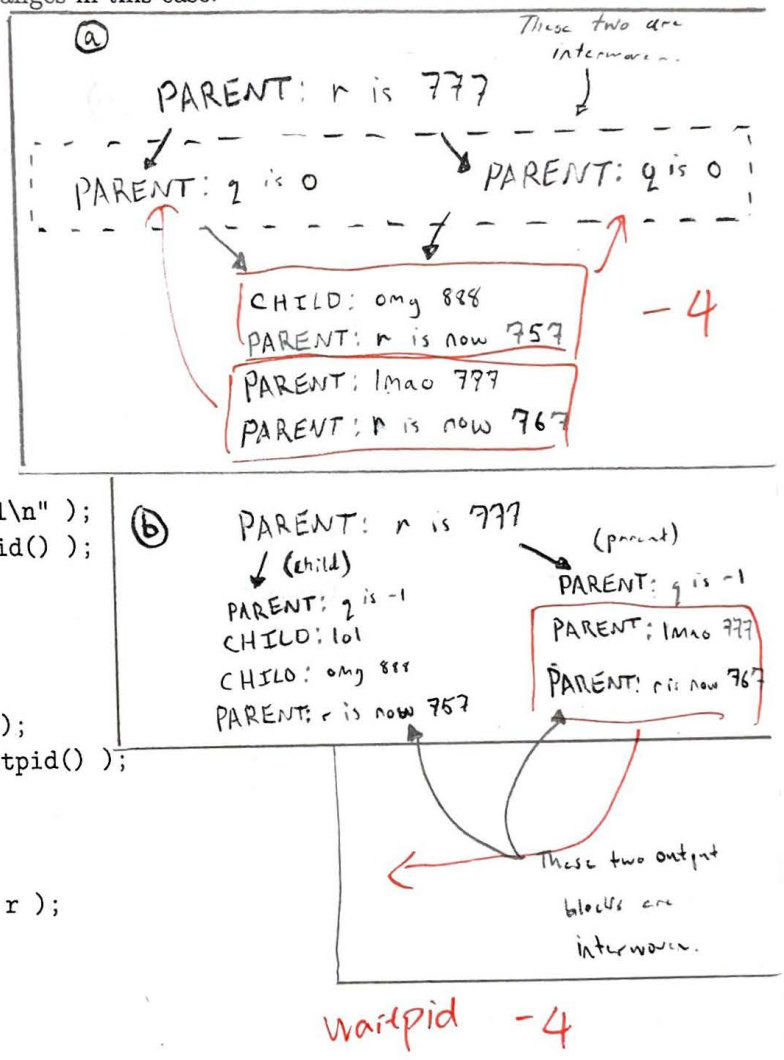
Estimated CPU Burst Time	Actual CPU Burst Time
$\tau_0 = 8$ ms	$t_0 = 12$ ms
$\tau_1 = 0.5(12) + 0.5(8)$ $= 10$	$t_1 = 10$ ms
$\tau_2 = 0.5(10) + 0.5(10)$ $= 10$	$t_2 = 10$ ms
$\tau_3 = 0.5(10) + 0.5(10)$ $= 10$	$t_3 = 6$ ms
$\tau_4 = 0.5(6) + 0.5(10)$ $= 8$	$t_4 = 9$ ms
$\tau_5 = 0.5(9) + 0.5(8)$ $= 8.5$	$t_5 = 10$ ms

4. **(23 POINTS)** Answer the following two questions given the C program below. If multiple outputs are possible, succinctly describe all possibilities by using a diagram. Be sure to describe how interleaving occurs, if applicable.

   (a) What is the **exact** terminal output of this code?

   (b) What is the **exact** terminal output of this code if variable q starts at 0 instead of 1? Only show how the output from (a) changes in this case.

```
int main()
{ int q = 0 // b
 //int q = 1;
  pid_t r = getpid();
  printf( "PARENT: r is %d\n", r );

  pid_t s = fork();
  q--;
  printf( "PARENT: q is %d\n", q );

  if ( s == 0 ) // child
  {
    if ( q < 0 ) printf( "CHILD: lol\n" );
    printf( "CHILD: omg %d\n", getpid() );
    r -= 20; // 777 - 20 = 757
  }
  else
  {
    if ( q < 0 ) waitpid( s, &q, 0 );
    printf( "PARENT: lmao %d\n", getpid() );
    r -= 10;
  }

  printf( "PARENT: r is now %d\n", r );

  return EXIT_SUCCESS;
}
```



5. **(23 POINTS)** Answer the following two questions given the C program below. If multiple outputs are possible, succinctly describe all possibilities by using a diagram. Be sure to describe how interleaving occurs, if applicable.

(a) What is the **exact** terminal output of this code?

(b) What is the **exact** contents of the `VV.txt` file?

```
int main()
{
  printf( "%dZZ", getpid() );
  fprintf( stderr, "%dYY", getpid() );
  close( 2 ); // Closes Stderr
  fprintf( stderr, "%dXX", getpid() );
  printf( "%dWW", getpid() );

  int fd = open( "VV.txt", O_WRONLY | O_CREAT | O_TRUNC, 0660 ); // fd = 2
  printf( "%dUU\n", getpid() );
  fprintf( stderr, "%dTT", fd );

  int rc = fork();

  if ( rc > 0 )
  {
    waitpid( rc, NULL, 0 );
  }
  else
  {
    printf( "%dSS\n", getpid() );
    fprintf( stderr, "%dRR\n", getpid() );
    close( fd ); // Closes "2" - Stderr
  }

  printf( "%dQQ\n", getpid() );
  fprintf( stderr, "%dPP\n", getpid() );

  return EXIT_SUCCESS;
}
```

(a)

777YY
777Z777WW — 6
777UU

(child)
888SS
888QQ

(parent)
777QQ

(b) VV.txt contents:

777UU — 6 — orig. process
888RR — child process
777PP — Parent process

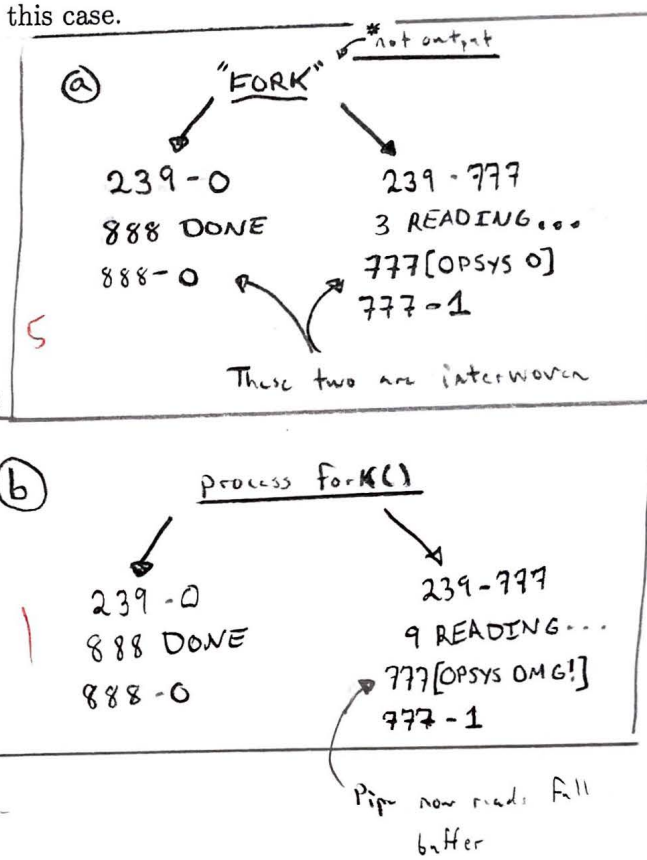
↑ file contents change 3 times

6. **(23 POINTS)** Answer the following two questions given the C program below. If multiple outputs are possible, succinctly describe all possibilities by using a diagram. Be sure to describe how interleaving occurs, if applicable.

   (a) What is the **exact** terminal output of this code? Assume `PART_B` is not defined.

   (b) What is the **exact** terminal output of this code if it is compiled using `-D PART_B`? Only show how the output from (a) changes in this case.

```
int main()
{
  close( 1 + 1 ); // close stderr
  int p[2];
  int q = 3 * 3;
  char buffer[4 * 4];
  int rc = pipe( p );
  pid_t r = fork();
  printf( "%d%d%d-%d\n", p[0], p[1], q, r );

#ifdef PART_B // runs w/ -D option
  q *= p[1]; // q = 9 * 3 = q = 27
#endif

  if ( r == 0 )
  {
    q -= 3; // q = 6
    r = write( p[1], "OPSYS OMG!", q );
    printf( "%d DONE\n", getpid() );
  }

  if ( r > 0 )
  {
    q = q / 3; // q = 3 | q = 1
    printf( "%d READING...\n", q );
    rc = read( p[0], buffer, q ); //
    buffer[rc] = '\0';
    printf( "%d[%s]\n", getpid(), buffer );
  }

  printf( "%d-%d\n", getpid(), r );

  return EXIT_SUCCESS;
}
```



7. **(15 POINTS)** For parts (a) and (b) below, calculate the individual turnaround and wait times for each of the processes described in the table below. Also count the number of preemptions each process experiences. Note that all "ties" are broken based on process ID order (i.e., alphabetical order). Ignore context switch times and other such overhead.

Process	CPU Burst Time	Arrival Time
A	9 ms	0 ms
B	5 ms	1 ms
C	4 ms	4 ms
D	2 ms	7 ms

(a) Apply the Shortest Job First (SJF) scheduling algorithm, filling in your results below.

Process	Turnaround Time	Wait Time	Number of Preemptions
A	9	0	0
B	19	14	0
C	11	7	0
D	4	2	0

SJF is non-preemptive

(b) Apply the Round Robin (RR) scheduling algorithm with a time slice of 3 ms, filling in your results below. As with Project 1, if a process is using the CPU and its time slice expires with no processes on the ready queue, do not count this as a preemption.

Process	Turnaround Time	Wait Time	Number of Preemptions
A	20 ms	11 ms	2
B	13 ms	8 ms	1
C	11 ms	7 ms	1
D	4 ms	2 ms	0

(c) Increasing the time slice from 3 milliseconds, at what point (i.e., what numeric time slice value) does the RR algorithm from (b) degenerate into the FCFS algorithm?

9 ms time slice

8. **(1 POINT EXTRA CREDIT)** Not including the original parent process, how many processes are created via the code below?

```
int main()
{
  int j = 4;
  while ( j > 0 )
  {
    pid_t p = fork();
    j--;
  }
  return EXIT_SUCCESS;
}
```

Parent
- j = 4
- j = 2
- j = 1

→ 3 processes are created

0

USE THE REST OF THIS PAGE FOR ADDITIONAL WORK.