1. **[25 POINTS]** For parts (a) through (d) below, calculate the individual turnaround times and total wait times for processes P₁, P₂, P₃, and P₄ as described in the following table:

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival time</th>
<th>CPU burst time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0 ms</td>
<td>12 ms</td>
</tr>
<tr>
<td>P₂</td>
<td>0 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>P₃</td>
<td>3 ms</td>
<td>5 ms</td>
</tr>
<tr>
<td>P₄</td>
<td>9 ms</td>
<td>3 ms</td>
</tr>
</tbody>
</table>

[0.5 pts for each answer in the tables; including "ms" is not required]

a. **[4 points]** Apply the First-Come First-Served (FCFS) scheduling algorithm.

<table>
<thead>
<tr>
<th>Process</th>
<th>Turnaround Time</th>
<th>Total Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>12 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>P₂</td>
<td>21 ms</td>
<td>12 ms</td>
</tr>
<tr>
<td>P₃</td>
<td>23 ms</td>
<td>18 ms</td>
</tr>
<tr>
<td>P₄</td>
<td>20 ms</td>
<td>17 ms</td>
</tr>
</tbody>
</table>

b. **[6 points]** Apply the Shortest Remaining Time (SRT) scheduling algorithm, i.e. a preemptive version of the Shortest Job First (SJF) algorithm. Also calculate the number of preemptions each process experiences.

<table>
<thead>
<tr>
<th>Process</th>
<th>Turnaround Time</th>
<th>Total Wait Time</th>
<th>Preemptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>29 ms</td>
<td>17 ms</td>
<td>0</td>
</tr>
<tr>
<td>P₂</td>
<td>17 ms</td>
<td>8 ms</td>
<td>2</td>
</tr>
<tr>
<td>P₃</td>
<td>5 ms</td>
<td>0 ms</td>
<td>0</td>
</tr>
<tr>
<td>P₄</td>
<td>3 ms</td>
<td>0 ms</td>
<td>0</td>
</tr>
</tbody>
</table>
c. **[6 points]** Apply the Round Robin (RR) scheduling algorithm with a time slice (i.e. quantum) of 2 milliseconds. Also calculate the number of preemptions each process experiences.

<table>
<thead>
<tr>
<th>Process</th>
<th>Turnaround Time</th>
<th>Total Wait Time</th>
<th>Preemptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>29 ms</td>
<td>17 ms</td>
<td>5</td>
</tr>
<tr>
<td>P₂</td>
<td>27 ms</td>
<td>18 ms</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>18 ms</td>
<td>13 ms</td>
<td>2</td>
</tr>
<tr>
<td>P₄</td>
<td>13 ms</td>
<td>10 ms</td>
<td>1</td>
</tr>
</tbody>
</table>

d. **[6 points]** Apply the Round Robin (RR) scheduling algorithm with a time slice (i.e. quantum) of 4 milliseconds. Also calculate the number of preemptions each process experiences.

<table>
<thead>
<tr>
<th>Process</th>
<th>Turnaround Time</th>
<th>Total Wait Time</th>
<th>Preemptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>28 ms</td>
<td>16 ms</td>
<td>2</td>
</tr>
<tr>
<td>P₂</td>
<td>29 ms</td>
<td>20 ms</td>
<td>2</td>
</tr>
<tr>
<td>P₃</td>
<td>21 ms</td>
<td>16 ms</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>14 ms</td>
<td>11 ms</td>
<td>0</td>
</tr>
</tbody>
</table>

e. **[3 points]** Increasing the time slice from 4 ms, at what point (i.e. what time slice value) does the RR algorithm degenerate into the FCFS algorithm?

12 ms (largest CPU burst time)
2. [25 POINTS] Given the following C program, answer the questions presented below. Assume that all system calls complete successfully. Further, assume that the parent process ID is 64, with any child processes numbered 128, 129, 130, etc.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid_t * x = (pid_t *)malloc( sizeof( pid_t ) );

    printf( "PARENT: %d\n", getpid() );
    pid = fork();
    printf( "PARENT: %d\n", pid );

    *x = pid * 10;
    if ( pid == 0 )
    {
        printf( "CHILD: !\n" );
        *x *= 5;
        printf( "CHILD: %d\n", *x );
    }
    else if ( pid > 0 )
    {
        printf( "PARENT: %d\n", *x );
    }

    *x -= 10;
    printf( "PARENT: (-: %d\n", *x );
    free( x );
    return 0;
}
```

a. [17 points] Write the exact terminal output in the space below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

![Diagram of possible outputs](image_url)

[1pt] PARENT: 64

PARENT outputs:
[2pts] PARENT: 128
[2pts] PARENT: 1280
[2pts] PARENT: (-: 1270

CHILD outputs:
[ ] PARENT: 0
[2pts] CHILD: !
[2pts] CHILD: 0
[2pts] PARENT: (-: -10

[2pts] these PARENT and CHILD output lines are interleaved
[2pts] the order of output lines in PARENT and CHILD must each be in relative orders shown

go on to the next page →
b. [8 points] If the processes in the code on the previous page were dispatched using the First-Come First-Served (FCFS) scheduling algorithm, write the exact terminal output in the space below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

[2pts] PARENT: 64
[1pt] PARENT: 128
[1pt] PARENT: 1280
[1pt] PARENT: (--: 1270

[ ] CHILD: 0
[1pt] CHILD: 1
[1pt] CHILD: 0
[1pt] PARENT: (--: -10

(i.e. no interleaving, because PARENT runs without interruption)
3. [25 POINTS] Given the following C program, answer the questions presented below. Assume that all system calls complete successfully. Further, assume that the parent process ID is 128, with any child processes numbered 256, 257, 258, etc.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/wait.h>

int main()
{
    close( 1 );

    int rc = open( "j.txt", O_WRONLY | O_CREAT | O_TRUNC );
    pid_t pid = fork();
    printf( "[? %d]", pid );

    if ( pid == 0 )
    {
        fprintf( stderr, "[?!]" );
        rc = dup2( 1, 2 );
        fprintf( stderr, "[&& %d]", rc );
    }
    else if ( pid > 0 )
    {
        wait( NULL );
        fprintf( stderr, "[$$ %d]", pid );
    }

    return EXIT_SUCCESS;
}
```

a. [9 points] Write the exact terminal output in the space below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

```
3pts
vvvv
[?!]$$ 256]

3pts 3pts
```
b. [9 points] Write the exact contents of the j.txt file below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

```
3pts
vvvvv
[& & 2] [? 0] [? 256]
^^^^^^     ^^^^^^^
3pts  3pts (last two segments appear in either order)
```

c. [7 points] If the wait() system call is omitted from the code on the previous page, write the exact terminal output in the space below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

```
3pts
vvvvv
[$$ 256] [?!]
^^^^     ^^^^  
2pts 2pts (segments appear in either order)
```
Given the following C program, answer questions (a) and (b) below. Assume that all system calls complete successfully. Further, assume that the parent process ID is 256, with any child processes numbered 512, 513, 514, etc.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

    printf( "%d-%d-%d\n", getpid(), p[0], p[1] );

    rc = fork();
    if ( rc == 0 )
    {
        rc = write( p[1], "DOIREALLYLOOKLIKEAGUYWITHAPLAN?", 31 );
    }
    if ( rc > 0 )
    {
        int n = p[0] * p[0] - p[1];
        char * buffer = (char *)calloc( n + 1, sizeof( char ) );

        rc = read( p[0], buffer, n );
        printf( "%d-%s\n", getpid(), buffer );

        free( buffer );
    }

    return EXIT_SUCCESS;
}
```

(a) **9 points** Write the **exact** terminal output in the space below. If multiple outputs are possible, please succinctly describe all possibilities by using a diagram.

<table>
<thead>
<tr>
<th>Points</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3pts]</td>
<td>256-3-4</td>
</tr>
<tr>
<td><strong>PARENT:</strong></td>
<td>256-DOIRE</td>
</tr>
<tr>
<td><strong>CHILD:</strong></td>
<td>512-ALLYL</td>
</tr>
<tr>
<td>[1pt]</td>
<td>PARENT and CHILD lines could be interleaved (swapped)</td>
</tr>
<tr>
<td>[1pt]</td>
<td>PARENT could be ALLYL instead of DOIRE and likewise for CHILD</td>
</tr>
</tbody>
</table>

(b) **4 points** Why is the first argument of the `calloc()` call `n+1` (instead of just `n`)?

<table>
<thead>
<tr>
<th>Points</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4pts]</td>
<td>Extra byte is for the '\0' character (NULL or EOS terminator)</td>
</tr>
</tbody>
</table>
c. **[12 points]** Given the following C program, write the exact terminal output in the space below. As per usual, if multiple outputs are possible, please describe all possibilities by using a diagram. Assume that all system calls complete successfully. Further, assume that the parent process ID is 512, with any child processes numbered 1024, 1025, 1026 etc.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    char b[20];
    int p[2];
    int rc = pipe( p );

    int pid = fork();

    if ( pid > 0 )
    {
        close( p[0] );
        rc = dup2( p[1], 1 );
    }

    printf( "0987654321" );
    fflush( NULL );

    if ( pid == 0 )
    {
        close( p[1] );
        rc = read( p[0], b, 6 );
        b[rc] = '\0';
        printf( "%d-%s\n", getpid(), b );
    }

    return EXIT_SUCCESS;
}
```

```
4pts
vvvvv
09876543211024-098765
^^^^^^^^^^    ^^^^^^  
4pts    4pts
```

go on to the next page →
5. **[EXTRA CREDIT +1]** In what year did Dr. Goldschmidt’s favorite breakfast cereal’s mascot first appear on cereal boxes?

[+1pt] 1965

6. **[EXTRA CREDIT +1]** What infamous villain hangs from Dr. Goldschmidt’s keychain (with feet that light up)?

[+1pt] The Joker

all done 😊