Sean Lossef losses@rpi.edu

1. **(4 POINTS)** What is the output when the following lines are run as part of a program? Circle the best answer.

```
char *z = "Hello World";
char *s = z;
int x = strlen(s) - strlen("World");
s[x] = '\0';
printf("[%s]\n",z);
```

(a) `[World]`

(b) `[Hello]`

(c) `[Hello ]`

(d) `[Hello World\n]`

(e) `[H]`

(f) No output, segmentation fault

2. **(4 POINTS)** How many **total** bytes are leaked by the following code? Circle the best answer.

```
int main()
{
  char * z = calloc( 32, sizeof( char ) );
  int pid = fork();
  if (pid==0)
  {
    char *x = malloc(32 * sizeof(*x));
  } else {
    waitpid(pid, NULL, 0);
    int **x = malloc(32 * sizeof(*x));
    free(z);
  }
  return EXIT_SUCCESS;
}
```

(a) 64

(b) 128

(c) 288

(d) 320

(e) 512

(f) No bytes are leaked

For Questions 3, 4 and 5, consider a virtual memory that uses static and equal allocation schemes. For given process `Q`, assume that the page reference string is as shown below.

`1 2 3 4 1 1 3 5 1 2 4 3`

3. **(5 POINTS)** Assume that all processes have four frames of physical memory, with the initial four page allocations shown below.

```
                 1  2  3  4  1  1  3  5  1  2  4  3
               ---------------------------------
                1  1  1  1
                .  2  2  2
                .  .  3  3
                .  .  .  4
               ---------------------------------
page faults ===>  p  p  p  p
```

Given the remaining eight page references shown above, how many page faults occur if the least-frequently-used page replacement algorithm is applied? Assume that the LRU page replacement algorithm is used to break any ties. Be sure to count the initial four page faults already shown. Circle the best answer.

(a) 7 page faults
(b) 8 page faults
(c) 9 page faults
(d) 10 page faults
(e) 5 page faults
(f) 6 page faults

4. **(5 POINTS)** Assume that all processes have four frames of physical memory, with the initial four page allocations shown below.

```
                 1  2  3  4  1  1  3  5  1  2  4  3
               ---------------------------------
                1  1  1  1
                .  2  2  2
                .  .  3  3
                .  .  .  4
               ---------------------------------
page faults ===>  p  p  p  p
```

Given the remaining eight page references shown above, how many page faults occur if the optimal page replacement algorithm is applied? Assume that the LRU page replacement algorithm is used to break any ties. Be sure to count the initial four page faults already shown. Circle the best answer.

(a) 7 page faults
(b) 8 page faults
(c) 9 page faults
(d) 10 page faults
(e) 5 page faults
(f) 6 page faults

5. **(5 POINTS)** Now assume that you want to go to a dynamic allocation and are attempting to determine a working set. At the point indicated, what is the working set if the working set delta is 4? Circle the best answer.

```
1  2  3  4  1  1  3  5  1  2  4  3
                      ^
                      |
```

(a) Working set is {5, 1, 2, 4}
(b) Working set is {5}
(c) Working set is {5, 1, 2, 4, 3}
(d) Working set is {1, 3, 5}
(e) Working set is {1, 3, 4, 5}
(f) None of these are correct

6. **(5 POINTS)** What is the **exact** output of the code snippet below? Circle the best answer.

```
char c = 'C';
unsigned char d = 0;
for (int ctr=0; ctr<512+c+10; ctr++) d++;
printf( "<%c>\n", d );
```

(a) `<77>`
(b) `<512>`
(c) `<525>`
(d) `<C>`
(e) `<M>`
(f) No output due to a segmentation fault

For questions 7 and 8, refer to the following code

```
int main()
{
char buf[32];
int p[2];
close( 0 );
close( 1 );
pipe( p );
dup2( 0, 3 );
dup2( 1, 4 );
open( "q7.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600 );
dup2( 2, 6 );
dup2( 3, 7 );
dup2( 4, 8 );
dup2( 5, 9 );
close( 3 );
close( 4 );
close( 5 );
write(8, "This is a line of text.", 18);
read(7, buf, 14);
write(1, "bicycle", 10);
write(9, "fishsticks", strlen("fishsticks"));
write(6, buf, 10);
read(p[0], buf, 32);
write(6, buf+4, 7);
/* ... */
}
```

7. **(5 POINTS)** After executing the code shown above, what is the output to the file q7.txt and how many file descriptors are still able to write the file? You can assume all data written to the file has been flushed. Circle the best answer.

   (a) bicycle (1 file descriptor)
   (b) fishstick (1 file descriptor)
   (c) bicycle (2 file descriptors)
   (d) fishstick (2 file descriptors)
   (e) bicycle (3 file descriptors)
   (f) fishstick (3 file descriptors)

8. **(5 POINTS)** After executing the code shown above, what is the output to the terminal and how many file descriptors are still able to write to the pipe? Circle the best answer.

   (a) This is a bicycle (2 file descriptors)
   (b) This is a fishstick (2 file descriptors)
   (c) This is a line of text.bicycle (2 file descriptors)
   (d) This is a bicycle (3 file descriptors)
   (e) This is a fishstick (3 file descriptors)
   (f) This is a line of text.bicycle (3 file descriptors)

9. **(5 POINTS)** The code below is an example of what? Circle the best answer.

```
int main()
{
size_t size = 2;
int *x = malloc(size * sizeof(int));
char val = 'c';
for (int ctr=0; ctr<(1<<15);ctr++)
{
if (ctr >= size)
{
size *= 2;
val++;
realloc(x, size * sizeof(int));
}
*(x+ctr) = val;
printf("%c\n", val);
}
return EXIT_SUCCESS;
```

(a) Buffer overflow
(b) A fork-bomb
(c) Uninitialized memory
(d) A segmentation fault
(e) An infinite loop
(f) A use-after-free (UAF) error

For questions 10 and 11, refer to the following code

```
#define THREADS 8
void * q10( void * arg )
{
  int *q = malloc(sizeof(int));
  *q = *((int *)arg);
  free(arg);
  *q += 2;
  pthread_exit((void *) q);
}
int main()
{
  pthread_t tid[8];
  for (int ctr=0; ctr<THREADS; ctr++)
  {
    int * x = malloc(sizeof(int));
    *x = ctr;
    pthread_create( &tid[ctr], NULL, q10, x );
  }
  for (int ctr=THREADS-1; ctr >=0; ctr--)
  {
    int *buf;
    pthread_join( tid[ctr], (void **) &buf );
    printf("%d", *buf);
    free(buf);
  }
  return EXIT_SUCCESS;
}
```

10. **(5 POINTS)** What output cannot occur from running this code? Circle the best answer.

(a) `99999999`
(b) `98624537`
(c) `11111111`
(d) `23456789`
(e) `All of these outputs are possible`
(f) `None of these outputs can occur`

11. **(5 POINTS)** How many different outputs are possible? Circle the best answer.

(a) `1`
(b) `2`
(c) `4`
(d) `8`
(e) `16`
(f) `32`

12. **(5 POINTS)** Consider a non-contiguous memory allocation scheme in which page tables for processes are stored in physical memory. Each memory access takes $110ns$. To speed up memory accesses, a translation look-aside buffer (TLB) is in place with an access time of $10ns$. What is the effective memory access time (EMAT) if the TLB hit ratio is 98%? Circle the best answer.

(a) $122.2ns$
(b) $95.8ns$
(c) $115.8ns$
(d) $230ns$
(e) $170.6ns$
(f) $112ns$

13. **(5 POINTS)** In a Linux filesystem with a block size of $Q = 4096$ bytes and inodes with 12 direct block pointers, what is the largest file that can be stored without using triple indirection? Assume that an indirect pointer block has a maximum of 1024 pointers. Circle the best answer.

(a) 4,402,345,721,856 bytes
(b) 5,499,210,752 bytes
(c) 4,299,210,752 bytes
(d) 4,243,456 bytes
(e) 61,440 bytes
(f) 49,152 bytes

14. **(5 POINTS)** Consider a non-contiguous memory allocation scheme in which a logical memory address is represented using 32 bits. Of these bits, the high-order 10 bits represents the page number; the remaining bits represent the page offset. What is the page number (in base 2) for the logical memory address below? Circle the best answer.

```
56,789,013    ==  11011000101000100000010101
          10                                2
```

(a) `1101100010`
(b) `1101`
(c) `11011000101000100000001`
(d) `10001010000100000010101`
(e) `0000010101`
(f) `10101`

15. **(8 POINTS)** How many processes are created for the code below? You should count the parent process in your total. Circle the best answer.

```
int main()
{
  close( 0 );
  int rc;
  int p[2];
  pipe( p );
  rc = fork();
  if ( rc == 0 )
  {
    rc = write(1, "Child\n", strlen("Child\n"));
  }
  if ( rc > 0 )
  {
    rc = fork();
    close(1);
  }
  return EXIT_SUCCESS;
}
```

(a) Exactly one process

(b) Exactly three processes

(c) Exactly two processes

(d) Exactly five processes

(e) Exactly four processes

(f) Exactly six processes

16. **(8 POINTS)** The code below will display exactly two lines of output. What line cannot appear for the **second** line of output? Assume all calls return successfully. Circle the best answer.

```
void * q16( void * arg )
{
  int * q = (int *)arg;
  *q -= 6;
  printf( "LUCKY %d\n", *q );
  return NULL;
}

int main()
{
  pthread_t tid1, tid2;
  int x = 16;
  int rc = pthread_create( &tid1, NULL, q16, &x );
  rc = pthread_create( &tid2, NULL, q16, &rc );
  x = 13;
  rc = pthread_join( tid1, NULL );
  rc = pthread_join( tid2, NULL );
  return EXIT_SUCCESS;
}
```

(a) `LUCKY -6`

(b) `LUCKY 7`

(c) `LUCKY 4`

(d) (a), (b) cannot appear

(e) (b) and (c) cannot appear

(f) All lines can appear

17. **(8 POINTS)** The server below has just started running. What is the **exact** terminal output of this code if a client connects via TCP and sends a packet containing the string "`Welcome`" and then a few minutes later sends a packet with the string "`To`', followed by the message "`Summertime`" (note no newline characters)? Assume all packets are received. Circle the best answer.

```
#define MAXBUFFER 6
int main()
{
  char buffer[ MAXBUFFER + 1 ];
  int sd = socket( PF_INET, SOCK_DGRAM, 0 );
  struct sockaddr_in server;
  server.sin_family = PF_INET;
  server.sin_addr.s_addr = htonl( INADDR_ANY );
  server.sin_port = htons( 8888 );
  bind( sd, (struct sockaddr *)&server, sizeof( server ) );
  struct sockaddr_in client;
  int i, n;
  socklen_t len = sizeof( client );
  do
  {
    n = recvfrom( sd, buffer, MAXBUFFER, 0, (struct sockaddr *) &client, &len );
    fprintf( stderr, "<" );
    for ( i = 0 ; i < n ; i += 2 ) fprintf( stderr, "%c", buffer[i] );
    fprintf( stderr, ">" );
  }
  while ( n > 0 );
  return EXIT_SUCCESS;
}
```

(a) `<Wlo><T><Sme>`

(b) `<Wloe><T><Smetm>`

(c) `<Welcome><To><Summertime>`

(d) `<Summertime>`

(e) `<ecm><o><umr>`

(f) No output

18. **(8 POINTS)** Given the set of processes described in the table below, which CPU scheduling algorithm produces the shortest wait time for process `P4`? All "ties" are broken based on process ID (i.e., increasing) order. Ignore context switch times and other such overhead. Circle the best answer.

Process	CPU Burst Time	Arrival Time
`P1`	9 ms	0 ms
`P2`	12 ms	0 ms
`P3`	6 ms	2 ms
`P4`	4 ms	5 ms

(a) RR with a time slice of 3 ms

(b) RR with a time slice of 5 ms

(c) SJF

(d) FCFS

(e) SRT

(f) Multiple algorithms produce the longest wait time for process `P4`