1. a. When statically linked libraries are used there must be a copy of it included in every executable file that uses that library while with dynamically linked libraries, they can share the same file, and they do not each need to have their own copy.
   Drawback is that the library has to be loaded when the program starts up, slowing down program startup.

   b. Floating point numbers are stored with:
      i. the sign bit
      ii. the significand (or mantissa)
      iii. the exponent
      i. The sign bit is 1 for positive numbers, 0 for negative.
      ii. The mantissa is normalized to an "implied leading 1" or the mantissa is $1 <= M < 2$.
      iii. The exponent is biased (always positive but meaning value–bias in order to get negative exponents)

   c. i. 00101001_2 in decimal is 41.

      ii. $10001111_2 + 00001011_2$ in binary is 10011010.

      iii. 11011011_2 in decimal is -37.

      iv. AB_{16} in decimal is 171 in decimal.

      v. $ABCD_{16}$ in binary is 1010101111001101.

      vi. 174_{10} in hexadecimal is AE.

   d. < is used to redirect input from a file to a program (change standard input) and > is used to redirect output from a program to file (change standard output).

   e. - They both pass one integer pointer as the one and only parameter.
      - `func1` says that the pointer itself is constant, and `func2` says that the integer pointed to by the pointer is constant.

   f. Loop unrolling is when the compiler duplicates the body of a loop a certain number of times, decreasing the number of iterations, so the effect of the modified loop is exactly the same. This is done to keep the pipeline full, which increases execution speed.

   g. The Pthreads function is `pthread_join`(), and the equivalent function for a process is **waitpid**().

2. 2 and 7
   Jones
   e and e
   f and s
   3 and 4

3. 5 11   0
    11   7 11
   ??? ??? 11

4. Here is one solution:

```
void my_system(char *string) {
  pid_t pid;
  char *args[40];  /* the problem gives this arbitrary number of arguments */
  int status;

  /* convert line into array of arguments */
  parse(string, args);

  pid= fork();

  if (pid == 0) {  /* child */

    if (execvp(args[0], args) == -1) {
      perror("execvp");
      exit(-1);
    }

  } else

    if (pid > 0)  /* parent */

      /* "wait(&status)" would also work, since only one child was created */
      waitpid(pid, &status, 0);

    else {
      printf("Error, unable to create a new process.\n");
      exit(-1);
    }
}
```

5. Variations of the assembly code could be written and have the same effect.

```
pushl   %ebp              # save old frame pointer in new stack frame
rrmovl  %esp, %ebp        # set new frame pointer
irmovl  $8, %eax          # sub 8 (pointer and int)
subl    %eax, %esp        #   from stack pointer
mrmovl  12(%ebp), %ebx    # load p parameter from actual parameter
rmmovl  %ebx, 4(%esp)     # store p into current stack frame
mrmovl  8(%ebp), %ecx     # load n parameter from actual parameter
rmmovl  %ecx, (%esp)      # store n into stack frame
irmovl  $4, %eax          #
multl   %ecx, %eax        # multiply n by sizeof(int)
addl    %ebx, %eax        # compute address of p[n]
mrmovl  (%eax), %eax        # load p[n]
wrint   %eax
rrmovl  %ebp, %esp        # reset stack pointer
popl    %ebp              # restore frame pointer
ret
```

6. Note the initialization of the semaphore, and that both functions use the **same** semaphore.

```
static sem_t mutex;

void init(void) {
  sem_init(&mutex, 0, 1);
}

int deposit(Account *account, unsigned int amount) {
  sem_wait(&mutex);
  account -> balance += amount;
  sem_post(&mutex);
  return account -> balance;
}

int withdraw(Account *account, unsigned int amount) {
  sem_wait(&mutex);
  account -> balance -= amount;
  sem_post(&mutex);
  return account -> balance;
}
```