3. **[20 Points]** Consider the following C program. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
pid_t pid;

void bar(int sig) {
  fprintf(stderr,"poke");
  exit(0);
}
void baz(int sig) {
  fprintf(stderr,"pen");
  exit(0);
}
void foo(int sig) {
  fprintf(stderr,"pig");
  kill(pid, SIGUSR1);
}
main() {
  signal(SIGUSR1, foo);
  signal(SIGCHLD, baz);
  pid = fork();
  sleep(1);
  if (pid == 0) {
    signal(SIGUSR1, bar);
    kill(getppid(), SIGUSR1);
    while(1) {};
  }
  else {
    pid_t p; int status;
    if ((p = wait(&status)) > 0) {
      fprintf(stderr,"farm");
    } else {
      fprintf(stderr,"ham");
    }
  }
}
```

What is the output string that this program prints?

2. **[20 Points]** You've been hired to optimize the a gaussian blur filter on the world's tiniest images, each of which are only 8×8. You start with the code:

```
char m[3][8][8] = .....;

for (int j = 1; j < cols-1; j++) {
  for (int i = 1; i < rows-1; i++) {
    for (int p = 0; p < 3; p++) {
      char up    = m[p][i-1][j];
      char down  = m[p][i+1][j];
      char left  = m[p][i][j-1];
      char right = m[p][i][j+1];
      .......   = (m[p][i][j] + left + right + up + down)/5;
    }
  }
}
```

You should assume: Char takes 1 bytes; you should ignore the store / memory writes – only consider loads ; The array 'm' starts at address 0; memory addresses are 12 bits long; All scalars are held in registers. Your cache is 2-way set associate with 4 byte lines, and a total size of 32 bytes and a "least recently used" replacement policy.

Below, list the address for the first 12 READ or LOAD references (ignore the Store/Write) and indicate if it is a hit or miss in the cache. Use decimal numbers throughout. It's easy to to first write down the array entry (*e.g.* `m[1][2[3]`), translate that to an address and then figure out the hit or miss.

[12 Points]

Ref #	Address	Array Entry	Hit?
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

[4 Points]

Below, draw a diagram to show the state of the cache at the end of the references above. You should clearly distinguish each set. For each cache line, you should indicate if the entry is valid and the appropriate **starting memory address** for that line/block (if valid). If the entry is not valid, just leave the data blank and/or have the tag be zero (we're ignoring the valid bit in this example). Rather than showing the "tag bits", which are harder to compute, indicate the *starting memory address of the cache block, which should be evenly divisible by the block size*. All numbers must be decimal.

[4 Points]

Assume the two loops were switched into this order:

```
char m[3][8][8] = .....;
for (int p = 0; p < 3; p++) {
  for (int i = 1; i < rows-1; i++) {
    for (int j = 1; j < cols-1; j++) {
       ....
    }
  }
}
```

How many cache misses would occur when that code is executed, assuming that's the only code that executed and the cache was initially empty.