Princeton University
COS 217: Introduction to Programming Systems
Fall 2005 Final Exam Answers

The exam was a three-hour, open-book, open-notes exam.

Question 1 (a)

```
Syntax error:  Invalid declaration statement on line 2.
Syntax error:  Unterminated comment on line 3.
Semantic error:  Incorrect computation on line 5.

int foo(int n) {
   int n2, n3;
     /* This function is busted */
   n2 = n * n;
   n3 = n2 * n;
   printf("n=%d, n^2 = %d, n^3 = %d\n", n, n2, n3);
   return 0;
}
```

Question 1 (b)

```
%s is a string is a string
```

Question 1 (c)

```
x = 12.3
i = 45
y = .6
```

Question 1 (d)

```
4
```

Question 1 (e)

```
110
```

Question 1 (f)

```
mystery(i) prints the value of i as a binary number.
```

Question 1 (g)

```
 65 (decimal) = 01000001 (binary)
 14 (decimal) = 00001110 (binary)
-14 (decimal) = 11110010 (binary)

  01000001
+ 11110010
----------
  00110011

00110011 (binary) = 51 (decimal)
```

Question 1 (h)

```
A virtual memory address consists of two parts:  a virtual page number and an offset.
The hardware uses the process's virtual page table to translate the virtual page number
into the page's start address in memory.  It then adds the offset to that page start
address to form the physical address.
```

Question 1 (i)

A function call executes in user mode. A system call executes in privileged (alias supervisor) mode.

A function call is executed via a call instruction. A system call is executed via an int (interrupt) instruction.

Question 1 (j)

The I/O functions declared in stdio.h do buffering for efficiency. Physically reading data from and writing data to devices is expensive. Buffering data in memory allows physical I/O to occur less frequently. Calling the fflush() function causes all data that is buffered for a specified output stream to be written to its physical device.

Question 1 (k)

The alarm() function causes a SIGALRM signal after a delay measured in *real* (alias *wall-clock*) time. The setitimer() function causes a SIGPROF signal after a delay measured in *virtual* (alias *CPU*, alias *system*) time.

Question 2

```
#define _GNU_SOURCE

/* Must define _GNU_SOURCE so the definition of the sigset_t type is included from
   signal.h.  With _GNU_SOURCE defined, a signal is not automatically reset to its
   default handler when it occurs.  So there is no need to call signal() near the end
   of the signal's handler.  */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <assert.h>
#include <unistd.h>

enum {MAX_SIGINT_COUNT = 2};
enum {MAX_SIGALRM_COUNT = 10};
enum {ALARM_SECONDS = 1};

static int iSigintCount = 0;

static void sigintHandler(int iSignal)
{
   iSigintCount++;
   if (iSigintCount == MAX_SIGINT_COUNT)
      exit(EXIT_SUCCESS);
}

static void sigalrmHandler(int iSignal)
{
   static int iSigalrmCount = MAX_SIGALRM_COUNT + 1;
   iSigalrmCount--;
   if (iSigalrmCount == 0)
   {
      printf("Happy New Year!\n");
      iSigalrmCount = MAX_SIGALRM_COUNT;
      iSigintCount = 0;
   }
   printf("%d... ", iSigalrmCount);
   fflush(stdout);
   alarm(ALARM_SECONDS);
}

int main(void)
{
   void (*pfRet)(int);
   int iRet;
   sigset_t sSigSet;
```

```
   /* Make sure SIGALRM is not blocked. */
   sigemptyset(&sSigSet);
   sigaddset(&sSigSet, SIGALRM);
   iRet = sigprocmask(SIG_UNBLOCK, &sSigSet, NULL);
   assert(iRet == 0);

   pfRet = signal(SIGALRM, sigalrmHandler);
   assert(pfRet != SIG_ERR);
   pfRet = signal(SIGINT, sigintHandler);
   assert(pfRet != SIG_ERR);

   alarm(ALARM_SECONDS);
   /* or raise(SIGALRM); */
   /* or kill(getpid(), SIGALRM); */

   for (;;)
      ;
}
```

Alternate Answer:

```
#define _GNU_SOURCE

/* Must define _GNU_SOURCE so the definition of the sigset_t type is included from
   signal.h.  With _GNU_SOURCE defined, a signal is not automatically reset to its
   default handler when it occurs.  So there is no need to call signal() near the end
   of the signal's handler.  */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <assert.h>
#include <unistd.h>

enum {MAX_SIGALRM_COUNT = 10};
enum {ALARM_SECONDS = 1};

static void sigintHandler(int iSignal)
{
   /* Reset SIGINT to its default handler. */
   signal(SIGINT, SIG_DFL);
}

static void sigalrmHandler(int iSignal)
{
   static int iSigalrmCount = MAX_SIGALRM_COUNT + 1;
   void (*pfRet)(int);

   iSigalrmCount--;
   if (iSigalrmCount == 0)
   {
      printf("Happy New Year!\n");
      iSigalrmCount = MAX_SIGALRM_COUNT;
      pfRet = signal(SIGINT, sigintHandler);
      assert(pfRet != SIG_ERR);
   }
   printf("%d... ", iSigalrmCount);
   fflush(stdout);

   alarm(ALARM_SECONDS);
}

int main(void)
{
   void (*pfRet)(int);
   int iRet;
   sigset_t sSigSet;

   /* Make sure SIGALRM is not blocked. */
```

```
   sigemptyset(&sSigSet);
   sigaddset(&sSigSet, SIGALRM);
   iRet = sigprocmask(SIG_UNBLOCK, &sSigSet, NULL);
   assert(iRet == 0);

   pfRet = signal(SIGALRM, sigalrmHandler);
   assert(pfRet != SIG_ERR);
   pfRet = signal(SIGINT, sigintHandler);
   assert(pfRet != SIG_ERR);

   alarm(ALARM_SECONDS);
   /* or raise(SIGALRM); */
   /* or kill(getpid(), SIGALRM); */

   for (;;)
      ;
}
```

Question 3 (a)

LDI and JNZ use *immediate* addressing. STORE, READ, and NANDM use *direct* addressing.

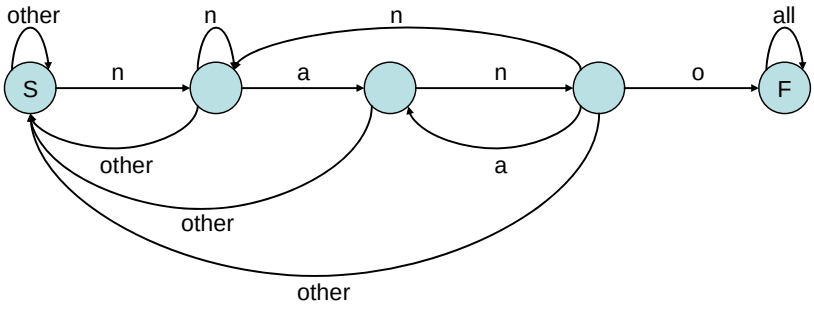
Question 3 (b)

```
READ (0x80)
SHR
STORE (0x80)
LDI 0x7f
NANDM (0x80)
LDI 0xff
NANDM (0x80)
```

Question 3 (c)

```
shift_left_start:
   LDI 0x01
   OUT
shift_left_continue:
   SHL
   OUT
   JNZ shift_left_continue
shift_right_start:
   LDI 0x80
   OUT
shift_right_continue:
   LSHR
   OUT
   JNZ shift_right_continue
   LDI shift_left_start
   JMP
```

Question 4



Question 5

```
$ make
gcc -c myprog.c
gcc -c one.c
gcc -c two.c
gcc -c three.c
gcc -c four.c
gcc -o myprog myprog.o one.o two.o three.o four.o

$ touch myprog.c
$ make
gcc -c myprog.c
gcc -o myprog myprog.o one.o two.o three.o four.o

$ make
make: Nothing to be done for `myprog'.

$ touch two.h
$ make
gcc -c two.c
gcc -o myprog myprog.o one.o two.o three.o four.o


$ touch four.h
$ make
gcc -c two.c
gcc -c three.c
gcc -c four.c
gcc -o myprog myprog.o one.o two.o three.o four.o
```

Question 6

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define MAX_CHARS_PER_LINE 256
#define MAX_LINES 10

int main(void)
{
   char pcLine[MAX_CHARS_PER_LINE];
   char *ppcLines[MAX_LINES];
   int i;

   for (i = 0; i < MAX_LINES; i++)
      ppcLines[i] = NULL;
```

```
   while (fgets(pcLine, MAX_CHARS_PER_LINE, stdin) != NULL)
   {
      free(ppcLines[0]);
      for (i = 0; i < MAX_LINES-1; i++)
         ppcLines[i] = ppcLines[i+1];
      ppcLines[MAX_LINES-1] = (char*)malloc(strlen(pcLine) + 1);
      assert(ppcLines[MAX_LINES-1] != NULL);
      strcpy(ppcLines[MAX_LINES-1], pcLine);
   }

   for (i = 0; i < MAX_LINES; i++)
      if (ppcLines[i] != NULL)
         fputs(ppcLines[i], stdout);

   for (i = 0; i < MAX_LINES; i++)
      free(ppcLines[i]);

   return 0;
}
```