CSC 407: Computer Systems II: Final (2017)

Joe Phillips
Last modified 2017 November 15

Name: ____

Distance Learning Students Only!

If you want your graded final returned to you please write your address below:

____

____

____

4 points free, then 16 points per question

1. **Optimization and Compilers**

   There are at least 4 optimizations that can be made in `optimizeMe()`. Find four optimization and for each:

   a. *do* it,
   b. tell whether the *compiler* or *programmer* should make it,
   c. tell *why* either the compiler or programmer (as opposed to the other) should make it

```
//  PURPOSE:  To return some integer value computed from 'arg0'.
extern
int             someFunction    (int    arg0
                                );
                                //  I will spare you the irrelevant details


//  PURPOSE:  To harass Computer Systems II students.  Computes some arbitrary
//      function of 'intArrayLen' and 'intArray' that I pulled out of my a**.
//      Returns its value.
int     optimizeMe              (int    intArrayLen0,   const int* intArray0,
                                 int    intArrayLen1,   const int* intArray1
                                )
{
  int   i0,i1;
  int   sum     = 0;

  for  (i0 = 0;  i0 < intArrayLen0;  i0++)
  {
    for  (i1 = 0;  i1 < intArrayLen1-1;  i1++)
    {
      if  (someFunction(intArray0[i0]) == 2*(intArray1[i1] + intArray1[i1+1]) )
        sum++;
    }
  }

  return(sum % 8);
}
```

Num	Optimitization (just do above)	Compiler or Programmer?	Why done by the person (or program) you said?
(i)			
(ii)			
(iii)			
(iv)			

2. **Memory**

Consider a process running the following program:

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

#define         TO_PRINT        "Good luck!"

const char*     toPrintCPtr     = TO_PRINT;

int             i               = 0;

int             main            ()
{
  for  (i = 0;  i < sizeof(TO_PRINT)-1;  i++)
    printf("%c %c\n",toPrintCPtr[i],toupper(toPrintCPtr[i]));

  return(EXIT_SUCCESS);
}
```

Please tell where the following objects are stored in memory.

Your choices are:

a. ROM BIOS
b. kernal Memory (the OS)
c. shared library memory (the glibc library)
d. .text segment
e. .rodata segment
f. .data segment
g. .bss segment
h. the heap
i. the stack

Where is:

1. (4 Points) the function `main()`?
2. (4 Points) the memory for variable 'i'?
3. (4 Points) the string `"%c %c\n"`?
4. (4 Points) the code that is given string `"G G\n"` and actually prints its pixels to the screen?

3. **Processes, Exceptions and Signals**

A parent takes its child to an ice cream parlor because the child *loves* to put interesting toppings on its ice cream. The child chooses toppings and sends its choices back to the parent using a *pipe*. The parent remembers the choices of the child in a buffer, until the parent gets annoyed (*i.e.* the buffer gets full). The parent tells the child to stop by sending it `SIGINT`.

Please finish the following program.

```
/*-------------------------------------------------------------------------*
 *---                                                                   ---*
 *---           crazyIceCream.c                                         ---*
 *---                                                                   ---*
 *-------------------------------------------------------------------------*/

#include
#include
#include                // For memset(), strlen()
#include                // For pipe(), usleep()
#include                // For sigaction()
#include                // For wait()

const int       BUFFER_LEN      = 128;

int             shouldContinue  = 1;

void            stopContinuing  (int     sigNum
                                )
{
  shouldContinue        = 0;
}


int     main            ()
{
  int   childToParent[2];
  pid_t childPid;

  srand(getpid());
  // (A) Initialize 'childToParent[]' as a pipe
  printf("Parent: \"What would you like on your ice cream, sweetie?\"\n");

  childPid      = 0;  // <--- (B) Replace 0 to make a child process

  if  (childPid == 0)
  {
    struct sigaction    act;
    const char*         cPtr;

    //  (C)  Install signal handler to do 'stopContinuing()' when receive 'SIGINT'

    printf("Child \"Okay, gimme ...\"\n");
    while  (shouldContinue)
    {
      switch  (rand() % 10)
      {
      case 0 :  cPtr = "peanuts, ";             break;
      case 1 :  cPtr = "caramel, ";             break;
      case 2 :  cPtr = "strawberries, ";        break;
      case 3 :  cPtr = "maraschino cherries, "; break;
      case 4 :  cPtr = "grilled onions, ";      break;
      case 5 :  cPtr = "salsa, ";               break;
      case 6 :  cPtr = "sprinkles, ";           break;
      case 7 :  cPtr = "chocolate chips, ";     break;
      case 8 :  cPtr = "mustard, ";             break;
      case 9 :  cPtr = "hot sauce, ";           break;
      }

      printf("Child \"%s\"\n",cPtr);
      //  (D) Send cPtr to parent (how many bytes?)
      usleep(1000);
    }

    printf("Child: \"Okay, now let me eat it!\"\n");
    exit(EXIT_SUCCESS);
  }

  char  toppingsBuffer[BUFFER_LEN];
  char  requestBuffer[BUFFER_LEN];
  char* bufferEndPtr    = toppingsBuffer;

  while  (1)
  {
    int numBytes;

    //  (E) Receive text from child and put into 'requestBuffer'.
    //      Also set 'numBytes' to the number of bytes received.

    if  ( (numBytes + (bufferEndPtr - toppingsBuffer)) >= BUFFER_LEN)
      break;

    requestBuffer[numBytes]     = '\0';
    printf("Parent \"%sand?\"\n",requestBuffer);

    memcpy(bufferEndPtr,requestBuffer,numBytes);
    bufferEndPtr        += numBytes;
    *bufferEndPtr        = '\0';
  }

  printf("Parent \"I have %sthat is MORE than enough!\"\n",toppingsBuffer);
  //  (F) Tell child to stop by sending it 'SIGINT'

  //  (G) Wait for child to actually stop.
  return(EXIT_SUCCESS);
}
```

Sample Output:

```
$ ./crazyIceCream
Parent: "What would you like on your ice cream, sweetie?"
Child "Okay, gimme ..."
Child "caramel, "
Parent "caramel, and?"
Child "salsa, "
Parent "salsa, and?"
Child "strawberries, "
Parent "strawberries, and?"
Child "grilled onions, "
Parent "grilled onions, and?"
Child "mustard, "
Parent "mustard, and?"
Child "grilled onions, "
Parent "grilled onions, and?"
Child "caramel, "
Parent "caramel, and?"
Child "grilled onions, "
Parent "grilled onions, and?"
Child "peanuts, "
Parent "peanuts, and?"
Child "strawberries, "
Parent "strawberries, and?"
Child "chocolate chips, "
Parent "I have caramel, salsa, strawberries, grilled onions, mustard, grilled on
ions, caramel, grilled onions, peanuts, strawberries, that is MORE than enough!"
Child: "Okay, now let me eat it!"
```

4. **Threads**

   a. (4 Points) Why is the following:

   ```
   while ( !object.isReady() )
     pthread_cond_wait(&cond,&mutexLock);
   ```

   a better idea than just:

   ```
   if ( !object.isReady() )
     pthread_cond_wait(&cond,&mutexLock);
   ```

   b. (4 Points) You are writing a simple server application. The server should wait for clients with `accept()` However, when a client comes, it should both handle that client with `handleClient()`, *and* go back to

`accept()` the next client. How would you solve this with threads?
Do not write code, but tell what parent and child threads should do.

c. (4 Points) Multiple threads need to access a *read-only* data-structure in memory. Does this data-structure need to be protected with **mutex locks**?
Why or why not?

d. (4 Points) To make a data-structure thread-safe, is it better to **put the locks and conditions in the methods of the data-structure** or to **make each thread do a `pthread_mutex_lock()` call before it calls a data-structure method and then do a `pthread_mutex_unlock()` after it finishes the call?**
Why?

5. **Practical C Programming**

a. (4 Points) Why should we use `snprintf()` instead of `sprintf()`, `strncpy()` instead of `strcpy()`, *etc.*? Seriously, how bad can using `sprintf()`, `strcpy()`, *etc.* be?

b. (4 Points) What does `extern` mean?
What does it tell the compiler to do?

c. (8 Points) The program below will compile well but run poorly. Please make it *do error checking* and fix it to make it proper:

```
#include        <stdlib.h>
#include        <stdio.h>
#include        <sys/types.h>   // for open()
#include        <sys/stat.h>    // for open()
#include        <fcntl.h>       // for open()

#define         BUFFER_LEN      256

int             main            (int   argc,
                                 char* argv[]
                                )
{
  char* filename        = argv[1];
  char  lookFor         = *argv[2];
  int   fd              = open(filename,O_RDONLY,0);
  int   count           = 0;
  char* buffer;
  int   numBytes;
  int   i;

  while  ( (numBytes = read(fd,buffer,BUFFER_LEN)) > 0)
    for  (i = 0;  i < numBytes;  i++)
      if  (buffer[i] == lookFor)
        count++;

  printf("%c was found %d times.\n",lookFor,count);
  return(EXIT_SUCCESS);
}
```

6. **Sockets and Files**

Finish the server function below which is told

- a minimum file size
- a maximum file size

It then iterates over the entries in the current directory (named `"."`) and returns the

- file length
- filename length
- filename

for every *file* (not directory or anything else) whose file length is between the minimum and maximum. The server tells the client it has no more by sending:

- `0` as the file length
- `0` the filename length
- no filename

All integers (min, max, file lengths and filename lengths) are sent in network endianness!

Example:

If a directory has the following files:

```
$ ls -lt
total 112
-rwxrw----. 1 instructor instructor  7388 Nov 14 14:23 client
-rw-rw-r--. 1 instructor instructor  2117 Nov 14 14:23 client.c
-rwxrw----. 1 instructor instructor 13760 Nov 14 14:17 server
-rw-rw-r--. 1 instructor instructor  2980 Nov 14 14:17 server.c
-rw-rw-r--. 1 instructor instructor 14406 Nov 14 13:38 20178-1Fal_CSC407_Final.html
-rw-rw-r--. 1 instructor instructor 14405 Nov 14 07:45 20178-1Fal_CSC407_Final.html~
-rw-rw----. 1 instructor instructor   608 Nov 14 07:45 bad.c
-rwxrw----. 1 instructor instructor  4999 Nov 14 07:45 bad
-rw-rw----. 1 instructor instructor   620 Nov 14 07:31 bad.c~
-rw-rw----. 1 instructor instructor   696 Nov 14 07:03 optimizeMe.c
-rwxrw----. 1 instructor instructor  7491 Nov 14 06:49 crazyIceCream
-rw-rw----. 1 instructor instructor  2484 Nov 14 06:49 crazyIceCream.c
-rw-rw----. 1 instructor instructor   271 Nov 14 06:29 memory.c
-rw-rw-r--. 1 instructor instructor  1702 Aug 16 11:52 client.c~
-rw-rw-r--. 1 instructor instructor  2279 Aug 16 11:48 server.c~
-rw-rw-r--. 1 instructor instructor   439 Aug 16 11:46 header.h
```

and I ask for every file between length 500 and 1000:

```
$ ./client
Machine name (e.g. localhost)? localhost
Port number? 2000
```

```
Please enter a minimum filesize: 500
Please enter a maximum filesize: 1000
```

then the program returns these 3 files:

```
                 bad.c~ 620
                  bad.c 608
           optimizeMe.c 696
```

Protocol:

```
        server                  client
          |                       |
          |  500 (network endian) |
          |<----------------------|
          |                       |
          | 1000 (network endian) |
          |<----------------------|
          |                       |
          |                       |
          |                       |
          |                       |
          |  620 (network endian) | (the length of bad.c~)
          |---------------------->|
          |                       |
          |    6 (network endian) | (the length of the string "bad.c~")
          |---------------------->|
          |                       |
          | "bad.c~"              | (do not send the quotes or the '\0' char)
          |---------------------->|
          |                       |
          |  608 (network endian) | (the length of bad.c)
          |---------------------->|
          |                       |
          |    5 (network endian) | (the length of the string "bad.c")
          |---------------------->|
          |                       |
          | "bad.c"               | (do not send the quotes or the '\0' char)
          |---------------------->|
          |                       |
          |  696 (network endian) | (the length of optimizeMe.c)
          |---------------------->|
          |                       |
          |    9 (network endian) | (the length of the string "optimizeMe.c")
          |---------------------->|
          |                       |
          | "optimizeMe.c"        | (do not send the quotes or the '\0' char)
          |---------------------->|
          |                       |
          |    0 (network endian) | (means "end of files")
          |---------------------->|
          |                       |
          |    0 (network endian) | (means "end of files")
          |---------------------->|
```

The function `handleClient(void* vPtr)` is run in its own thread. It receives `vPtr` which points an integer file descriptor for talking to the client. It should:

A. Cast `vPtr` to type `int*` and set `clientFd` to the integer passed
B. `free()` pointer `vPtr`.
C. Get the value of `min` from the client. Then, change it from network endian to host (this computer) endian.
D. Get the value of `max` from the client. Then, change it from network endian to host (this computer) endian.
E. Set `dirPtr` to read from the current directory `"."`.
F. In a loop, set `entryPtr` equal to the address of the next entry read from `dirPtr`.
G. Fill `statBuffer` full of the meta-data about the current entry.
H. Look for only those entries that are files of the desired size (done for you)
I. Set `fileLen_net` and `fileNameLen_net` to the lengths of the file and of the filename *in network endian!*
J. Send `fileLen_net`, `fileNameLen_net` and the first `fileNameLen` bytes of the filename back to the client.
K. Close `dirPtr`
L. Set `fileLen_net` and `fileNameLen_net` both to `0` *in network endian!*
M. Send `fileLen_net` and `fileNameLen_net`back to the client.
N. Close `clientFd` (done for you)

Do not worry about error checking!

```
#define BUFFER_LEN      256

void*   handleClient    (void*  vPtr)
{
  char          buffer[BUFFER_LEN];
  int           clientFd       = 0; // (A) <-- change that 0
  unsigned int  min;
  unsigned int  max;

  // (B)


  // (C)


  // (D)


  unsigned int          fileLen_net;
  unsigned int          fileNameLen;
  unsigned int          fileNameLen_net;
  struct stat           statBuffer;
  struct dirent*        entryPtr;
  DIR*                  dirPtr          = NULL; // (E) <-- change that 0

  while  ( (entryPtr = /* (F) */ ) != NULL )
  {
    //  (G)


    if  ( //  (H)
          S_ISREG(statBuffer.st_mode)  &&
          (statBuffer.st_size >= min)  &&
          (statBuffer.st_size <= max)
        )
    {
      fileNameLen       = strlen(entryPtr->d_name);
      fileLen_net       = // (I0)
      fileNameLen_net   = // (I1)

      // (J)
    }

  }

  // (K)

  // (L)

  // (M)

  // (N)
  close(clientFd);
  return(NULL);
}
```