CSC 407: Computer Systems II: Final (2014 Spring)

Joe Phillips
Last modified 2014 June 9

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 should make it

```
//  PURPOSE:  To sort 'intArray[]' of length 'intArrayLen'.  The result is
//      identical to that if 'inefficientBubbleSort()' were called, but this
//      one is expected to be faster.  No return value.
void    efficientQuickSort      (int* intArray, int intArrayLen);
  //  I'll spare you the irrelevant details.


//  PURPOSE:  To sort 'intArray[]' of length 'intArrayLen'.  The result is
//      identical to that if 'efficientQuickSort()' were called, but this one
//      is expected to be slower.  No return value.
void    inefficientBubbleSort   (int* intArray, int intArrayLen);
  //  I'll spare you the irrelevant details.


//  PURPOSE:  To return the number of time the number '7' appears in
//      'intArray[]' of length 'intArrayLen'.  Doesn't change 'intArray' at all.
int     countNum7s      (const int* intArray, int intArrayLen);
  //  I'll spare you the irrelevant details.


// PURPOSE:  To harass CSC-407 students.  Computes some arbitrary function
//      I pulled out of my a**.  Returns its value.
int     optimizeMe      (const int* intArray, int intArrayLen)
{
  int   smallerSize  = intArrayLen/2;
  int*  smallerArray = (int*)calloc(sizeof(int),smallerSize);
```

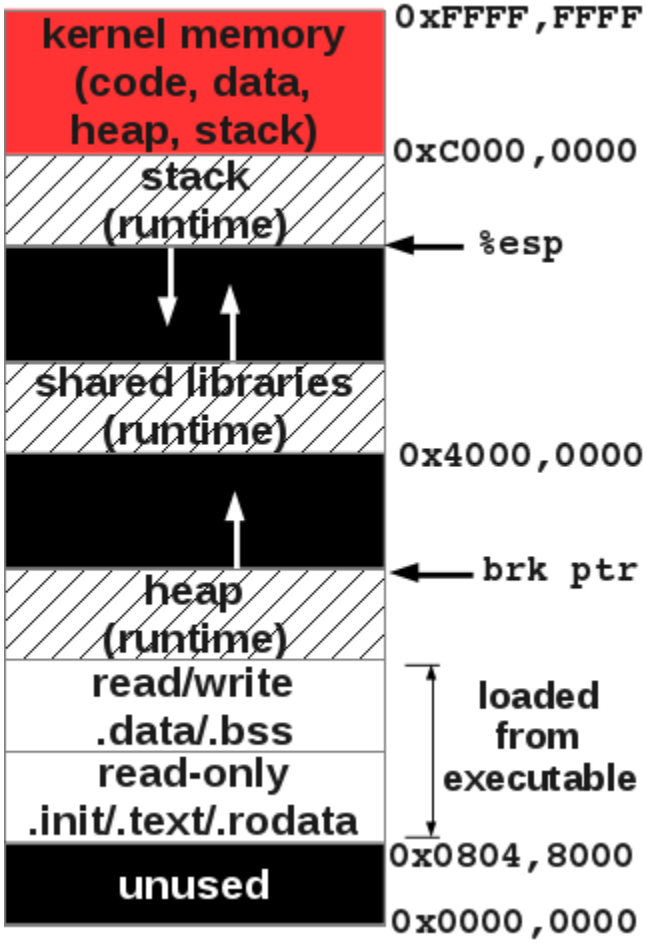
```
  int   sum          = 0;

  for  (int i = 0;  i < smallerSize;  i++)
    smallerArray[i] = intArray[i*2];

  inefficientBubbleSort(smallerArray,smallerSize);

  for  (int i = 0;  i < smallerSize;  i++)
    sum += (countNum7s(smallerArray,smallerSize) - smallerArray[i]);

  free(smallerArray);
  return(sum);
}
```

Num	Optimitization (just do above)	Compiler or Programmer?	Why done by the person (or program) you said?
(a)			
(b)			
(c)			
(d)			

2. **Memory**

A running program has:

- 0x0004,0000 bytes of code, string constants and other read-only memory loaded from its executable file.
- 0x0004,0000 bytes of global variables initialized to values other than 0.
- 0x0004,0000 bytes of global variables initialized to 0.
- used 0x0080,0000 bytes for objects allocated with `malloc()`, `new`, *etc*.
- used 0x0800,0000 byted for local variables, parameters, function return values, *etc*.
- dynamicaly linked with 0x0800,0000 bytes of shared library



HINTS:

- Compute the boundaries like where the global vars begin, heap begins, *etc*
- Remember this is *hexadecimal:* 0x800 + 0x800 = 0x1000

a. **How big was the executable file that was loaded from the disk?** (Tell any assumptions you made)
b. **What is the value of the `brk` pointer?**
c. **What is the value of the `%esp` register?**
d. **What is the top (highest address) in the memory used for shared libraries?**

3. **Processes, Exceptions and Signals**

a. (4 Points) A parent process `fork()`s a child process. Both processes want to send a sequence of integers to each other. Can this easily be done with *pipes*? Explain how it can, or why it cannot.

b. (4 Points) A parent process `fork()`s a child process. Both processes want to send a sequence of integers to each other. Can this easily be done with *signals*? Explain how it can, or why it cannot.

c. (4 Points) Let us say you write a program to measure how quick a person's fingers are by trapping `SIGINT` and then asking them to press `Ctrl-C` as rapidly as possible. The `SIGINT` signal handler increments a global counter every time `Ctrl-C` is typed. After a predefined time it stops and prints the global counter divided by the time used.
What is a fundamental problem with this program?

d. (4 Points) What is a zombie process?
Are zombie processes bad because they use a lot of CPU time?
How can a good programmer avoid having too many zombie processes?

4. **Threads**

Below is a C++ class for a buffer of integers. `void putIn (int i)` is used to put integers into it. `int getOut ()` is used to get integers from it.

One *teeny, tiny* problem with it . . . ***it's NOT thread safe!***

A. Make it *thread safe* by (A2) initializing, (A3) releasing, (A4&A5) using in `void putIn (int i)`, (A6&A7) using in `int getOut ()`, a member variable that you declare (A1).
B. Make it *usable* by having `void putIn (int i)` block when the buffer is full (use `bool isFull()`), and by having `int getOut ()` block when the buffer is empty (use `bool isEmpty()`). Your code should (B2) initialize, (B3) release, (B4&B5) using in `void putIn (int i)`, (B6&B7) using in `int getOut ()`, member variable(s) that you declare (B1).

```
const int       SIZE    = 16;

class   Buffer
{
  //  Member vars declared here:

  //  PURPOSE:  To hold the integers in the Buffer.
  int           array_[SIZE];
```

```
  //  PURPOSE:  To tell the next index to which to place an integer.
  int           inIndex_;

  //  PURPOSE:  To tell the next index from which to get an integer.
  int           outIndex_;

  //  PURPOSE:  To tell the number of integers in Buffer.
  int           numItems_;

  //  PURPOSE:  To make the Buffer thread safe.
  //  (A1)

  //  PURPOSE:  To make the Buffer usable.
  //  (B1)

public :
  //  PURPOSE:  To initialize an empty Buffer.  No parameters.  No return value.
  Buffer        ()
  {
    outIndex_ = inIndex_ = 0;
    numItems_ = 0;
    //  (A2)
    //  (B2)
  }


  //  PURPOSE:  To release resources.  No parameters.  No return value.
  ~Buffer       ()
  {
    //  (A3)
    //  (B3)
  }


  //  PURPOSE:  To return 'true' if Buffer is full or 'false' otherwise.
  //    No parameters.
  bool          isFull  () const
  {
    return(numItems_ == SIZE);
  }


  //  PURPOSE:  To return 'true' if Buffer is empty or 'false' otherwise.
  //    No parameters.
  bool          isEmpty () const
  {
    return(numItems_ == 0);
  }


  //  PURPOSE:  To place 'i' in Buffer.  No return value.
  void          putIn   (int    i)
  {
    //  (A4)
    //  (B4)

    array_[inIndex_] = i;
    inIndex_++;

    if  (inIndex_ >= SIZE)
      inIndex_ = 0;

    numItems_++;

    //  (B5)
```

```
    //  (A5)
  }


  //  PURPOSE:  To return an integer from the Buffer.  No parameters.
  int           getOut  ()
  {
    //  (A6)
    //  (B6)

    int i = array_[outIndex_];
    outIndex_++;

    if  (outIndex_ >= SIZE)
      outIndex_ = 0;

    numItems_--;

    //  (B7)
    //  (A7)

    return(i);
  }

};
```

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        <string.h>

#define         LINE_LEN        1024
#define         COMMENT_CHAR    '#'

int     main    (int    argc,
                 char*  argv[]
                )
{

  const char*   filename        = argv[1];
  FILE*         fp              = fopen(filename,"r");

  char* line;
  int   counter = 0;
  int   lineNum = 0;

  while  (fgets(line,LINE_LEN,fp) != NULL)
  {
    lineNum++;

    char*       cPtr;
```

```
    for  (cPtr = line;  *cPtr != '\0';  cPtr++)
      if  ( !isspace(*cPtr) )
        break;

    if  ( (*cPtr == COMMENT_CHAR) || (*cPtr == '\0') )
      continue;

    if  ( !isdigit(*cPtr) )
    {
      fprintf(stderr,"Mal-formed line at %d\n",lineNum);
      continue;
    }

    int i,j;

    if  (sscanf(cPtr,"%d %d",&i,&j) == 2)
      counter++;
    else
      fprintf(stderr,"Mal-formed line at %d\n",lineNum);
  }

  printf("%d\n",counter);
  return(EXIT_SUCCESS);
}
```

6. **Sockets and ncurses**

A client program already has a socket to a server program and is communicating using file descriptor `socketFD`. Write a function that:

- turns on ncurses
- turns off line buffering
- clears the screen
- turns off echoing
- In a loop, until the user types `QUIT_CHAR`:
  - Gets a character from the user's keyboard (without waiting for Enter)
  - Prints the character the user typed
  - Sends the character across the socket to the server
  - Gets one character from the server
  - Prints the character that the server sent
- After the user types `QUIT_CHAR` the function:
  - closes the socket,
  - waits 5 seconds,
  - turns ncurses off, and
  - returns.

```
void    printServersChars       (int    socketFD)
{

  //  YOUR CODE HERE

}
```

This particular server changes the case of the character sent to it. I typed "*abcdefghijk*"


instructor@JoesLaptopFedora16:~/Documents/Academic/DePaul/Classes...
File Edit View Search Terminal Help
aAbBcCdDeEfFgGhHiIjJkK