Computer Science Portfolio

A Portfolio
Presented to
the Computer Science Faculty of
Western Carolina University

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science
Computer Science

---

by
Joseph Randall Hunt
May 2013

Accepted by:
Dr. Kreahling – Systems

Abstract

This portfolio is a collection of the work I've done in Computer Science while attending WCU. I'm writing this so that prospective employers might see examples of my work and have a good understanding of what I'm capable of. Contained within this profile are examples of my work from CS 350 and CS 370. To date I have done several low level assignments for the Systems SLO. At WCU I've learned several things so far including: the importance of well engineered software; the ethics behind programming; the privacy laws and intellectual property laws programmers should be aware of; what is essential for system programming; how low level systems work; and finally the strengths of linux tools and how to use them.

Consent Form

I, Joseph Randall Hunt, hereby give the Faculty of the Computer Science Program at Western Carolina University permission to maintain, indefinitely, a copy of any student portfolio that I develop as part of my course work as an undergraduate in that program and to use those copies of my student portfolios for program assessment.

Signature

Date

Table of Contents

List of Listings

Chapter 1

Introduction

This portfolio was created to display my skills as a computer scientist and to showcase some of my work. In it you will find several examples of programs and descriptions of what I have learned. I believe this portfolio provides a good introduction to my abilities and what I have learned at Western.

1.1 Portfolio Organization

The remainder of this portfolio is organized as follows. Chapter 2 presents the student learning objectives on which this portfolio focuses. These objectives currently include: Systems. Expect this section to expand as I progress through my major. Chapter 3 contains a summary of what this portfolio included with some concluding remarks.

Chapter 2

Student Learning Objectives

This chapter consists of two major sections. Section 2.1 focuses on the technical skills you have learned during my time here at Western. Section 2.2 focuses on the skills I have acquired that are non-technical. The social skills section mainly focuses on my ethics course and my technical skills section focuses mainly on CS350.

2.1 Technical Skills

By "technical skills" I mean not just programming languages, but paradigms, algorithms, and engineering methodologies that differentiate a good programmer from a hobbyist. Included in this section are three SLOs: Software Engineering, Algorithms, and Systems. Though at this time I've completed only Systems. The other sections will be filled as I progress through my major.

2.1.1 Systems

It is vital for a computer scientist to understand how computer systems operate at the lowest level because it allows them not only to create better code but to create better systems. When one understands what their code is ultimately doing they can fine-tune it to perform much faster and more efficiently. In this section I have two work samples of MIPS assembly code and one example of C code demonstrating threading.

2.1.1.1 AtoI

This program shows my basic understanding of and ability to manipulate assembly code. It shows how to use system calls and how to go about allocating data for a program.While simple I feel it's necessary to include this because of it's relative importance to my overall portfolio. Demonstrating mastery of a language, while not necessarily inline with this SLOs goals is nevertheless very important. You can see this program in Appendix A.

2.1.1.2 Three Procedures

For this assignment we were required to write three procedures, and a driver, in MIPS. The procedures were `strlen`, `strncpy`, and `mem_align`. This program shows my understanding of low-level procedure calls: what needs to be passed, how, and what needs to be returned and how. Perhaps more importantly it shows what all needs to be kept track of when maintaining the executing environment across procedure calls, the stack pointer, the return address pointer, and so on.

Listing 2.1.1.2 shows how to go about manipulating the stack to store the registers according to the rules one follows when one invokes a procedure.

```
1  addi $sp, $sp, -16          # $sp-=4; /* allocate 4 words */
2  sw $s0, 0($sp)              # *($sp + 0) = $s0 /* save $s0 */
3  sw $s1, 4($sp)              # *($sp + 4) = $s1 /* save $s1 */
4  sw $s2, 8($sp)              # *($sp + 8) = $s2 /* save $s2 */
5  sw $s3, 12($sp)             # *($sp + 12) = $s3 /* save $s3 */
6  ------------------------------------------------------------
7  lw $s0, 0($sp)              # $s3 = *($sp + 0); /* restore $s0 */
8  lw $s1, 4($sp)              # $s3 = *($sp + 4); /* restore $s1 */
9  lw $s2, 8($sp)              # $s3 = *($sp + 8); /* restore $s2 */
10 lw $s3, 12($sp)             # $s3 = *($sp + 12); /* restore $s3 */
11 addi $sp, $sp, 16           # $sp+=4; /* deallocate 4 words */
```

You can see this entire program in Appendix B.

2.1.2 BoundedBuffer

This program was written for an assignment in CS370, Operating Systems. This program is most succinctly described as a thread-safe bounded circular buffer. Making use of the *POSIX Threading* library this program provides a block of memory to which threads can write and read. If a thread attempts to write to the buffer, and the space available in the buffer is sufficient to hold the data the thread is writing, the thread will copy the data to the buffer and continue executing. However if the buffer is full on a write, or empty on a read, then the thread will block waiting for a read() or write() call to be made. Listing 2.1.2 shows the prototype of the bounded buffer as provided by Dr. Dalton.

```
1  #ifndef BOUNDEDBUFFER_H
2  #define BOUNDEDBUFFER_H
3  #include <pthread.h>
4  typedef struct {
5    char* data;
6    int capacity;
7    int start;
8    int end;
9    int size;
10   pthread_mutex_t mutex;
11   pthread_cond_t cond;
12 } BoundedBuffer;
13 void bufferInit(BoundedBuffer* buffer, int capacity);
14 void bufferWrite(BoundedBuffer* buffer, char* data, int count);
15 void bufferRead(BoundedBuffer* buffer, char* data, int count);
16 void bufferDeallocate(BoundedBuffer* buffer);
17 #endif
```

You can see the implementation of these methods in Appendix C.

2.1.2.1 Reflection

I believe that an understanding of systems is very important for a computer scientist to quote the textbook for CS350, *Computer Organization And Design*, “The performance of software systems is dramatically affected by how well software designers understand the basic hardware

technologies at work in a system" [1]. In addition to that it is necessary for a computer scientist to understand the relationships between the software and hardware, the design choices, and the reasons for those design choices.

Reflecting on my work from this area of computer science I can safely say I have a strong grasp of systems concepts. I understand pipelining, multi-cycle data paths, caches, virtual memory, assembly language, computer arithmetic, memory hierarchies, and I/O from the low level side. I do however wish I had a more complete understanding of caches. While I understand how they work and what they do, I don't understand the specifics of how a cache is set up.

On a higher level Andrew S. Tanenbaum's book *Operating Systems: Design and Implementation*, the textbook for CS370, provides an excellent description of what someone dealing with higher level OS code should be proficient in: "system calls, processes, IPC, scheduling, I/O, deadlocks, memory management, threads, filesystems, and more" [2]. I believe I am proficient in all of these, and more. I had the benefit of learning C at a much higher level than I had ever dealt with before, shown in my third work sample from this SLO, which taught me quite a bit about programming. In addition dealing with MINIX in CS370 also taught me quite a bit about the low level code in an operating system. I am still lacking in some areas at the higher level though. I don't feel I have a firm understanding of DMA and I still don't entirely understand batch processing algorithms.

Despite the several pitfalls mentioned above I believe that my understanding of this topic is comprehensive.

2.2 Social Skills

By "social skills" I mean the ability to communicate effectively both in a technical setting and a non technical setting. This section includes three SLOs: Communication, Teamwork, and Ethics. Though at this time I've only completed Ethics. The other sections will be filled as I progress through my major.

Chapter 3

Conclusion

While this document is by no means complete it provides a good starting point for a profile that needs to be completed for my computer science degree. To summarize the content thus-far I can say that I have a good grasp of low level systems and the essential programming abstractions involved in those systems. What does this mean to a prospective employer? It means my code will be to a standard above that of the average code-monkey. I'll understand exactly how it is executing and behaving all the way down to the CPU. It means that I can work with new microprocessors and architectures with relatively little training.

My time at Western, though brief so far, has been one of the most informative experiences of my life. One thing I have learned is that a computer scientist's skills do not lay solely in the technical realm. While I can do the math and solve the programming problems, I believe the most essential skill for a computer scientist is the ability to understand complex multi-tiered abstractions and the ability to pick up knowledge from any area of study very quickly. A computer scientist is not just a machine that churns out code. A computer scientist is a person deserving of their title "scientist": they know how to research, they know basic science from several fields, and they're fully cognizant of how their program will interact or contribute to the overall product or experiment. So, as a prospective employer, you should ask yourself if all of your candidates posses the same non-technical skills that I do.

Appendices

Appendix A

MIPS AtoI

```
1  #Joseph Randall Hunt
2  #8/2/10
3  # without shifts or ands
4  # t0, address of buf
5  # t1, the individual bytes of buf
6  # t2, 48
7  # t3, int2ascii
8      .data
9  str: .asciiz "Please enter an integer > "
10 buf: .byte 0 0 0 0 0     # buffer for string, 4 values + nl
11 nl:  .byte 10 0          # newline, null
12 scs: .byte 32 58 32 0    # space colon space null
13 int2ascii: .byte 48      # add to int to get a char
14     .text
15 main:
16     li $v0, 4            # Print string syscall
17     la $a0, str          # load argument for syscall
18     syscall
19
20     li $a0, 0            # clear a0
21
22     li $v0, 8            # load read string syscall
23     la $a0, buf          # give the address of buf to a0
24     la $a1, 5            # give the length of buf+null to a1
25     syscall
26     ######### First Int #########
27     la $t0, buf          # load the address of buf into a0
28     lb $t1, ($t0)        # load the first byte of buf into t1
29
30     li $v0, 1            # print_int syscall
31     la $a0, ($t1)        # load address of int stored in t1 into a0
32     syscall
33
34     la $t3, int2ascii    # load address of 48 into t3
35     lb $t2, ($t3)        # put 48 into t2
36     sub $t1, $t1, $t2    # subtract 48 from char value to get int value
37
38     li $v0, 4
```

```
39      la $a0, scs         # print space colon space
40      syscall
41
42      li $v0, 1           # Print int syscall
43      la $a0, ($t1)       # Print the string
44      syscall
45
46      li $v0, 4
47      la $a0, nl          # print new line
48      syscall
49      ######## Second Int #########
50      lb $t1, 1($t0) #load second byte of buf into t1
51      li $v0, 1
52      la $a0, ($t1)
53      syscall
54      sub $t1, $t1, $t2
55      li $v0, 4
56      la $a0, scs
57      syscall
58      li $v0, 1
59      la $a0, ($t1)
60      syscall
61      li $v0, 4
62      la $a0, nl
63      syscall
64      ######### Third Int #########
65      lb $t1, 2($t0)      # load the third byte of buf into t1
66      li $v0, 1
67      la $a0, ($t1)
68      syscall
69      sub $t1, $t1, $t2
70      li $v0, 4
71      la $a0, scs
72      syscall
73      li $v0, 1
74      la $a0, ($t1)
75      syscall
76      li $v0, 4
77      la $a0, nl
78      syscall
79      ######### Last Int ##########
80      lb $t1, 3($t0)  # load the fourth byte of buf into t1
81      li $v0, 1
82      la $a0, ($t1)
83      syscall
84      sub $t1, $t1, $t2
85      li $v0, 4
86      la $a0, scs
87      syscall
88      li $v0, 1
89      la $a0, ($t1)
90      syscall
91      li $v0, 4
92      la $a0, nl
93      syscall
94      ######## EXIT ########
95      li $v0, 10          # Exit
```

```
96      syscall
```

Listing A.1: MIPS Assembly AtoI

Appendix B

MIPS Three Procedures

```
1  # Author: Joseph Randall Hunt
2  # Version: 2/20/10
3  #-------- Data Segment ---------
4      .data
5  prompt:             .asciiz "Please enter a string > "
6  test1:              .asciiz "Test 1:\nThe length of the string \""
7  test2:              .asciiz "Test 2:\nThe new String is: "
8  test3:              .asciiz "Test 3:\nFive: "
9  iscolonspace:       .asciiz "\" is: "
10 sixcolonspace:      .asciiz "\nSix: "
11 sevencolonspace:    .asciiz "\nSeven: "
12 eightcolonspace:    .asciiz "\nEight: "
13 string:             .space 256
14 stringcopy:         .space 256
15 nl:                 .byte 10 0
16 #-------- Text Segment ---------
17
18 ########## Nicely done: 15.9/16
19     .text
20     .globl main
21 main:
22     li $v0, 4                   # Print prompt
23     la $a0, prompt
24     syscall
25
26     li $v0, 8                   # Read string into string
27     la $a0, string
28     li $a1, 256
29     syscall
30     la $a0, nl                  # Print new line
31     li $v0, 4
32     syscall
33     ###############################################################################
34     ######## Registers #########
35     ##      s0 = strlen       ##
36     ##      s1 = straddr      ##
37     ##      s2 = string[-1]   ##
38     ##      s3 = compare      ##
```

```
39      ##                          ##
40      ############################
41      li $v0, 4                   # Print "Test 1:\n..."
42      la $a0, test1
43      syscall
44
45      la $a0, string              # Load the address of the string as an argument
46      li $a1, 256                 # Load the length of the array as an argument
47      jal strlen
48      move $s0, $v0               # Move the return of strlen into s0
49
50      addi $s0, $s0, -1           # subtract one from length
51      la $s1, string
52      add $s1, $s1, $s0
53      lb $s2, 0($s1)
54      li $s3, 10
55      bne $s3, $s2, skipremovenl
56      sb $zero, 0($s1)    ## nice
57  skipremovenl:
58      li $v0, 4                   # Print string
59      la $a0, string
60      syscall
61
62      la $a0, iscolonspace        # Print "\" is: "
63      li $v0, 4
64      syscall
65
66      move $a0, $s0
67      li $v0, 1                   # Load print_int syscall
68      syscall
69
70      la $a0, nl                  # Print new line
71      li $v0, 4
72      syscall
73      ###############################################################################
74      la $a0, nl                  # Print new line
75      li $v0, 4
76      syscall
77
78      li $v0, 4                   # Print "Test 2:..."
79      la $a0, test2
80      syscall
81
82      la $a0, stringcopy
83      la $a1, string
84      move $a3, $s0               # set max to strlen-1 (so no new line)
85      jal strncpy                 # jump to strncpy
86
87      la $a0, stringcopy          # Print the copied string
88      li $v0, 4
89      syscall
90
91      la $a0, nl                  # Print new line
92      li $v0, 4
93      syscall
94      ###############################################################################
95      la $a0, nl                  # Print new line
96      li $v0, 4
97      syscall
```

```
98      li $v0, 4                   # Print "Test 3: "
99      la $a0, test3
100     syscall
101     addi $a0, $zero, 5
102     jal mem_align
103     move $a0, $v0
104     li $v0, 1
105     syscall
106
107     la $a0, sixcolonspace
108     li $v0, 4
109     syscall
110     addi $a0, $zero, 6
111     jal mem_align
112     move $a0, $v0
113     li $v0, 1
114     syscall
115
116     la $a0, sevencolonspace
117     li $v0, 4
118     syscall
119     addi $a0, $zero, 7
120     jal mem_align
121     move $a0, $v0
122     li $v0, 1
123     syscall
124
125     la $a0, eightcolonspace
126     li $v0, 4
127     syscall
128     addi $a0, $zero, 8
129     jal mem_align
130     move $a0, $v0
131     li $v0, 1
132     syscall
133
134     la $a0, nl                  # Print new line
135     li $v0, 4
136     syscall
137     j exit
138     ###############################################################################
139 strlen:
140     #------- Registers -------#
141     #       s0=string[i]      #
142     #-------------------------#
143
144     addi $sp, $sp, -4           # $sp--; /* allocate 1 word */
145     sw $s0, 0($sp)              # *($sp + 0) = $s0 /* save $s0 */
146     add $s0, $zero, $zero       # clear s0
147     add $v0, $zero, $zero       # clear v0
148 strlenloop:
149 ######## line wrap (-.1)
150     lb $s0, 0($a0)              # $s0 = string[i] //Q: could somehow move out of loop?
151     beq $s0, $zero, strlenend   # exit if byte is null character
152     addi $v0, $v0, 1            # length++
153     addi $a0, $a0, 1            # i++... //Q: is there a way to only add once?
154     ble $v0, $a1, strlenloop    # loop if neccesary
155 strlenend:
156     lw $s0, 0($sp)              # $s1 = *($sp + 0); /* restore $s1 */
```

```
157     addi $sp, $sp, 4            # $sp++; /* deallocate 1 words */
158     jr $ra                      # return;
159     ##############################################################################
160
161 strncpy:
162     #------- Registers -------#
163     #   s0=i                  #
164     #   s1=&string[i]         #
165     #   s2=string[i]          #
166     #   s3=&strincopy[i]      #
167     #-------------------------#
168     addi $sp, $sp, -16          # $sp-=4; /* allocate 4 words */
169     sw $s0, 0($sp)              # *($sp + 0) = $s0 /* save $s0 */
170     sw $s1, 4($sp)              # *($sp + 4) = $s1 /* save $s1 */
171     sw $s2, 8($sp)              # *($sp + 8) = $s2 /* save $s2 */
172     sw $s3, 12($sp)             # *($sp + 12) = $s3 /* save $s3 */
173     add $s0, $zero, $zero       # Clear all saved registers
174     add $s1, $zero, $zero
175     add $s2, $zero, $zero
176     add $s3, $zero, $zero
177 L1:
178     add $s1,$s0,$a1             # $s1 = &string[i] in $s1
179     lb $s2, 0($s1)              # $s2 = string[i]
180     add $s3,$s0,$a0             # $s3 = &stringcopy[i]
181     sb $s2, 0($s3)              # stringcopy[i] = string[i]
182     beq $s2, $zero, strncpyend  # if (string[i] == 0) { goto strncpyend }
183     bge $s0, $a3, strncpyend    # if (i >= max)
184     addi $s0, $s0, 1            # i++
185     j L1                        # loop
186
187 strncpyend:
188     lw $s0, 0($sp)              # $s3 = *($sp + 0); /* restore $s0 */
189     lw $s1, 4($sp)              # $s3 = *($sp + 4); /* restore $s1 */
190     lw $s2, 8($sp)              # $s3 = *($sp + 8); /* restore $s2 */
191     lw $s3, 12($sp)             # $s3 = *($sp + 12); /* restore $s3 */
192     addi $sp, $sp, 16           # $sp+=4; /* deallocate 4 words */
193     jr $ra                      # return;
194     ##############################################################################
195
196 mem_align:
197     #------- Registers -------#
198     #   s0=quorem             #
199     #   s1=rem                #
200     #-------------------------#
201     addi $sp, $sp, -4
202     sw $s0, 0($sp)
203     sw $s1, 4($sp)
204     addi $s0, $zero, 4          # clear s0 then put 4 in it
205     add $v0, $zero, $zero       # clear v0
206     div $a0, $s0                # divide argument by 4
207     mfhi $s1                    # get remainder and put it in s1
208     beq $s1, $zero, remis0      # if s1 == 0 set v0 to 4 otherwise:
209     move $v0, $s1               # ret = rem
210 remret:
211     lw $s0, 0($sp)              # dealloc
212     lw $s1, 4($sp)              # dealloc
213     addi $sp, $sp, 4            # dealloc
214     jr $ra
215 remis0:
```

```
216     li $v0, 4
217     j remret
218     ################################################################################
219 exit:
220     li $v0, 10
221     syscall
```

Listing B.1: MIPS Assembly Three Procedures

Appendix C

Thread-safe Bounded Circular Buffer

```
1  #ifndef BOUNDEDBUFFER_H
2  #define BOUNDEDBUFFER_H
3  #include <pthread.h>
4
5  typedef struct {
6    char* data;
7    int capacity;
8    int start;
9    int end;
10   int size;
11   pthread_mutex_t mutex;
12   pthread_cond_t cond;
13 } BoundedBuffer;
14
15 void bufferInit(BoundedBuffer* buffer, int capacity);
16 void bufferWrite(BoundedBuffer* buffer, char* data, int count);
17 void bufferRead(BoundedBuffer* buffer, char* data, int count);
18 void bufferDeallocate(BoundedBuffer* buffer);
19
20 #endif
```

Listing C.1: BoundedBuffer.h

```
1  #include "BoundedBuffer.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void bufferInit(BoundedBuffer* buffer, int capacity) {
6    buffer->data = calloc(capacity, sizeof(char));
7    buffer->capacity = capacity;
8    buffer->start = 0;
9    buffer->end = 0;
10   buffer->size = 0;
11   pthread_mutex_init(&buffer->mutex, NULL);
12   pthread_cond_init(&buffer->cond, NULL);
13 }
14
15 void bufferWrite(BoundedBuffer* buffer, char* data, int count) {
16   pthread_mutex_lock(&buffer->mutex);
```

```
17   while (count > (buffer->capacity - buffer->size)) {
18     pthread_cond_wait(&buffer->cond, &buffer->mutex);
19   }
20   int i;
21   if (buffer->start == -1) { buffer->start = buffer->end; }
22   for (i = 0; i < count; i++) {
23     buffer->data[(i+buffer->end) % buffer->capacity] = data[i];
24     buffer->end = (buffer->end + 1) % buffer->capacity;
25     buffer->size++;
26   }
27   if (buffer->start == -1) { buffer->start = buffer->end; }
28   pthread_mutex_unlock(&buffer->mutex);
29   pthread_cond_broadcast(&buffer->cond);
30 }
31
32 void bufferRead(BoundedBuffer* buffer, char* data, int count) {
33   pthread_mutex_lock(&buffer->mutex);
34   while(count > (buffer->capacity - buffer->size))  {
35     pthread_cond_wait(&buffer->cond, &buffer->mutex);
36   }
37   int i;
38   if (buffer->start == buffer->end) { buffer->start = -1; }
39   for (i = 0; i < count; i++) {
40     data[i] = buffer->data[(i+buffer->start) % buffer->capacity];
41     buffer->start = (buffer->start + 1) % buffer->capacity;
42     buffer->size--;
43   }
44   if (buffer->start == buffer->end) { buffer->start = -1; }
45   pthread_mutex_unlock(&buffer->mutex);
46   pthread_cond_broadcast(&buffer->cond);
47 }
48
49 void bufferDeallocate(BoundedBuffer* buffer) {
50   pthread_mutex_lock(&buffer->mutex);
51   free(buffer->data);
52   pthread_mutex_unlock(&buffer->mutex);
53   pthread_mutex_destroy(&buffer->mutex);
54   pthread_cond_destroy(&buffer->cond);
55 }
```

Listing C.2: BoundedBuffer.c

Bibliography

[1] David A. Patterson and John L. Hennessy. *Computer Organization & Design: The Hardware/-Software Interface*. Morgan Kaufmann Publishers, San Francisco, California, 1994.

[2] Andrew S. Tanenbaum and Albert S. Woodhull. *Operating Systems Design and Implementation*. Pearson, Upper Saddle River, NJ, 3. edition, 2008.