CS33 Homework Assignment 5 Solutions

Fall 2019

1. We're accustomed to stacks that seem to be able grow without bound. But suppose we're running in an environment in which, while there's plenty of memory, it's not necessarily available in large contiguous pieces. Thus it might be necessary to have a segmented stack. At various points in the execution of a program, it's determined that the current stack is about to be exhausted, so we arrange so that the next function call takes place on a different stack. In particular, we assume the existence of the function *runonstack*:

   ```
   int runonstack(int (*func)(int), int arg, long *stackp);
   ```

   It calls *func*, passing it the argument *arg*, but arranges for *func* to execute on the stack that begins at *stackp* (i.e., if the stack is considered an array of longs, *stackp* is the address of the last (highest addressed) long in the array). *Runonstack* returns the *int* returned by *func*.

   Your task is to implement *runonstack* in x86-64 assembler code. You might use the following code to test your implementation:

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

int runonstack(int(*func)(int), int arg, long *stack);

int test(int a) {
      printf("in test %d\n", a);

      if (--a > 0) {
           printf("returned %d\n", test(a));
      }

      return a;
}

long stack[1024];

int main(int argc, char *argv[]) {
      if (argc != 2) {
           fprintf(stderr, "Usage: test count\n");
           exit(1);
      }

      int count = atoi(argv[1]);

      int ret = runonstack(test, count, &stack[1023]);
```

```
	printf("runonstack returned %d\n", ret);

	return 0;
}
```

Assuming your assembler code is in the file *runonstack.s* and the test program is in *test.c*, you can create an executable called *test* by doing:

```
gcc –o test runonstack.s text.c –O1 -g
```

The intended implementation of *runonstack* has ten lines. It should start with the following code (which comprise two of the ten lines), so that your code is given a name that can be referenced by your C code:

```
      .globl runonstack
runonstack:
```

The remaining lines are all assembler instructions. Note that if *func*, the address of the function that's passed to *runonstack*, is copied to %rax, then to call the function one uses the instruction

```
call *%rax
```

Answer:

```
      .globl runonstack
runonstack:
      movq  %rsp, (%rdx)     # save old stack pointer in new stack
      movq  %rdx, %rsp       # switch to new stack
      subq  $8, %rsp         # account for old sp being on the new stack
      movq  %rdi, %rax       # get address of func
      movl  %esi, %edi       # get func's arg and put it in appropriate reg
      call  *%rax            # call func
      movq  8(%rsp), %rsp    # restore old stack pointer
      ret                    # return to original caller
```