Benjamin Pierce (bgp12)
EECS 338
HW2
Due: 2/1/1

1)
a)
```
$ time ./for
Running for loop...

real	0m16.660s
user	0m16.660s
sys	0m0.000s
```

The for program does not invoke any system calls beside the single printf() at the very beginning, so the runtime is monopolized by user time, which is the repeating for loops.

b)
Program: for1b.c
```c
int main() {
	int i, j, k, N = 2;
	for(i = 0; i < N; i++) {
		for(j = 0; j < N; j++) {
			for(k = 0; k < N; k++) {
				sleep(1);
			}
		}
	}
}
```

```
$ time  ./for

real	0m8.003s
user	0m0.002s
sys	0m0.000s
```

By adding a 1 sec sleep, we reduce the time the program spends actively doing something, which reduces the sys and user time relative to the real time.

c) Program: for.c
```c
int main() {
	int i, j, k, N = 200;
	for(i = 0; i < N; i++) {
		for(j = 0; j < N; j++) {
			for(k = 0; k < N; k++) {
				printf("I = %d, J = %d, K = %d \n", i,j,k);
			}
		}
	}
}
```

```
real    0m31.262s
user    0m3.745s
sys     0m13.689s
```

By calling printf(), we increase the sys time, as printf() is a system call. Since so much printing is being done relative to actual looping, the system time is far greater

d)

```
top - 15:01:17 up 1 day,  2:26,  2 users,  load average: 0.07, 0.22, 0.24
Tasks: 249 total,   3 running, 179 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12.5 us,  4.2 sy,  0.0 ni, 82.9 id,  0.1 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem :  7985248 total,   387928 free,  2448984 used,  5148336 buff/cache
KiB Swap:     1020 total,     1020 free,        0 used.  5041244 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3638 bp0017    20   0  581484  45784  29796 D  93.3  0.6   0:36.21 xfce4-term+
10907 bp0017    20   0    4508    720    656 R  66.7  0.0   0:00.10 for
 1331 bp0017    20   0 1949660  11896   8628 S   6.7  0.1   1:29.26 pulseaudio
10908 bp0017    20   0   41776   3760   3124 R   6.7  0.0   0:00.01 top
  302 bp0017    20   0   22828   5436   3600 S   0.0  0.1   0:00.30 bash
  331 bp0017    20   0 98.837g 147612  69316 S   0.0  1.8   0:05.67 atril
  361 bp0017    20   0  187764   4272   3904 S   0.0  0.1   0:00.00 atrild
…
…
```

top claims we're using 66.7% CPU

From part c, we know that

```
real    0m31.262s
user    0m3.745s
sys     0m13.689s
```

and the formula for %CPU is (user + sys) / real

%CPU = (3.745+13.689)/31.262 = 55.8%CPU, which is relatively close to the value provided by top.

---

2)

First time running

```
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ ./for &
[1] 29370
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ pmap 29370
29370:   ./for
00005644c1351000      4K r-x-- for
00005644c1551000      4K r---- for
00005644c1552000      4K rw--- for
00007ff1b66d0000   1948K r-x-- libc-2.27.so
00007ff1b68b7000   2048K ----- libc-2.27.so
00007ff1b6ab7000     16K r---- libc-2.27.so
```

```
00007ff1b6abb000      8K rw--- libc-2.27.so
00007ff1b6abd000     16K rw---   [ anon ]
00007ff1b6ac1000    156K r-x-- ld-2.27.so
00007ff1b6cb6000      8K rw---   [ anon ]
00007ff1b6ce8000      4K r---- ld-2.27.so
00007ff1b6ce9000      4K rw--- ld-2.27.so
00007ff1b6cea000      4K rw---   [ anon ]
00007ffcc946f000    132K rw---   [ stack ]
00007ffcc9512000     12K r----   [ anon ]
00007ffcc9515000      8K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 total             4380K
```

Second Time Running

```
[1]+  Done                    ./for
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ ./for &
[1] 29402
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ pmap 29402
29402:   ./for
00005623ce1c7000      4K r-x-- for
00005623ce3c7000      4K r---- for
00005623ce3c8000      4K rw--- for
00007ff9ef25d000   1948K r-x-- libc-2.27.so
00007ff9ef444000   2048K ----- libc-2.27.so
00007ff9ef644000     16K r---- libc-2.27.so
00007ff9ef648000      8K rw--- libc-2.27.so
00007ff9ef64a000     16K rw---   [ anon ]
00007ff9ef64e000    156K r-x-- ld-2.27.so
00007ff9ef843000      8K rw---   [ anon ]
00007ff9ef875000      4K r---- ld-2.27.so
00007ff9ef876000      4K rw--- ld-2.27.so
00007ff9ef877000      4K rw---   [ anon ]
00007fff11923000    132K rw---   [ stack ]
00007fff119bf000     12K r----   [ anon ]
00007fff119c2000      8K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 total             4380K
```

a) The only address the two (and all C programs on my system) share in common is the last, which seems to be associated with vsyscall, which is a system call accelerator.
b) All the rest have different memory address, which is due to C deciding where in user memory it wants to place everything.
c) The C standard library files all have the same size, which makes sense because we did not alter them or the program between runs. The stack space for the variables is also the same, for the same reasons. In fact, it appears the complier kept everything the same size between runs of the exact same program; this behavior makes logical sense and is something we would expect.

---

3)

Program: address.c

```
#include <stdio.h>

int main(){
	int a = 0; // Regular integer
	const int b = 0; // Constant integer
	// Print the pointer values (addresses) in hexadecimal
	printf("%p, %p\n", &a, &b);
	sleep(30)
}
```

Result:

```
$ ./address
0x7fffd0a19380, 0x7fffd0a19384
```

a) Top output:

```
top - 18:17:25 up 17 min,  2 users,  load average: 0.13, 0.23, 0.23
Tasks: 232 total,   1 running, 166 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.5 us,  0.7 sy,  0.1 ni, 96.6 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7985248 total,  4670224 free,  1538684 used,  1776340 buff/cache
KiB Swap:     1020 total,     1020 free,        0 used.  6041052 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 6482 bp0017    20   0   41776   3672   3080 R  12.5  0.0   0:00.02 top
 1293 bp0017    20   0  191552  26240  19172 S   6.2  0.3   0:05.74 xfwm4
 1162 bp0017    20   0   76996   7956   6564 S   0.0  0.1   0:00.04 systemd
….
….
 6356 bp0017    20   0    4508    744    680 S   0.0  0.0   0:00.00 address
```

pmap output

```
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ ./address &
[1] 6694
0x7ffe7e51b6a0, 0x7ffe7e51b6a4
bp0017@bp0017-XPS-15-9550:~/Documents/EECS338/HW2$ pmap 6694
6694:   ./address
00005590c1cf6000      4K r-x-- address
00005590c1ef6000      4K r---- address
00005590c1ef7000      4K rw--- address
00005590c2387000    132K rw---   [ anon ]
00007fb38fe69000   1948K r-x-- libc-2.27.so
00007fb390050000   2048K ----- libc-2.27.so
00007fb390250000     16K r---- libc-2.27.so
00007fb390254000      8K rw--- libc-2.27.so
00007fb390256000     16K rw---   [ anon ]
00007fb39025a000    156K r-x-- ld-2.27.so
00007fb39044f000      8K rw---   [ anon ]
00007fb390481000      4K r---- ld-2.27.so
00007fb390482000      4K rw--- ld-2.27.so
```

00007fb390483000   4K rw---   [anon]
00007ffe7e4fd000   132K rw---   [stack]
00007ffe7e589000   12K r----   [anon]
00007ffe7e58c000   8K r-x--   [anon]
ffffffffff600000   4K r-x--   [anon]
 total   4512K

top claims that our program uses 4508K of memory. Pmap says that we use 4512K. They appear to agree within reasonable error.

b)
$./address
0x7fff0b354e90, 0x7fff0b354e94

pmap 8241
8241:   ./address
000055ee95374000   4K r-x-- address
000055ee95574000   4K r---- address
000055ee95575000   4K rw--- address
000055ee95843000   132K rw---   [anon]
00007f9b4d6e6000   1948K r-x-- libc-2.27.so
00007f9b4d8cd000   2048K ----- libc-2.27.so
00007f9b4dacd000   16K r---- libc-2.27.so
00007f9b4dad1000   8K rw--- libc-2.27.so
00007f9b4dad3000   16K rw---   [anon]
00007f9b4dad7000   156K r-x-- ld-2.27.so
00007f9b4dccc000   8K rw---   [anon]
00007f9b4dcfe000   4K r---- ld-2.27.so
00007f9b4dcff000   4K rw--- ld-2.27.so
00007f9b4dd00000   4K rw---   [anon]
00007fff0b335000   132K rw---   [stack]
00007fff0b3d6000   12K r----   [anon]
00007fff0b3d9000   8K r-x--   [anon]
ffffffffff600000   4K r-x--   [anon]
 total   4512K

The regular variable (left) is stored in the section of the memory known as the **stack**, since it can be alerted during runtime. The constant variable (right) is stored in the data section, since it only needs to be read and not altered. The permissions for the stack variables are read/write, which makes sense since we want to read and write to these variables during the execution of the program. Since we only need to read, it’s likely that the const variable is in 0x00007fff0b3d6000 due to that section being read-only.

c) The maximum hex address for the memory segment is 0x00007f7c01248000 + 8K bytes. 8K = 8192 bytes, which is 0x2000, so the max address is 0x7F7C0124A000. The max address for a 4 byte integer would be 0x00007f7c01248000 + 0x4 , which is 0x00007f7c01248004.

----

4)
Program: forker.c

```
#include <stdio.h>
#include <unistd.h> //to remove sleep warning
#include <sys/wait.h>

int main(){
	int pid = fork();
	if (pid){ //parent
		printf("I'm the parent! Waiting for child...\n");
		int childID = wait(NULL);
		printf("The PID of my child was %d \n",childID);
	}
	else{
		printf("I'm the child! My parent's PID is %d \n",getppid());
		sleep(20);
	}


}
```

Program output:

```
$ ./forker
I'm the parent! Waiting for child...
I'm the child! My parent's PID is 24715
The PID of my child was 24716
```

Top output

```
 PID  PPID   UID USER     RUSER    TTY          TIME+  %CPU %MEM S COMMAND
24716 24715  1000 bp0017   bp0017   pts/0      0:00.00   0.0  0.0 S forker
24715  1512  1000 bp0017   bp0017   pts/0      0:00.00   0.0  0.0 S forker
```

The two appear to agree.

----

5)
a)Two child processes are created. This makes sense, since fork is called twice in the same scope. We create a child, and then that child spawns another child, which begins execution from the second fork statement.
b) Yes, it is possible to have two child processes. For example, you could have a program that calls fork, and sets the child to do some form of work. Before the child is done, the program can fork again, and the two processes with the same parent can run concurrently. As in problem5c.c, it is also possible for the child to have a child, and the two can also run concurrently.
c) No, the program behaves differently. In this case, only one child is spawned, and then the child is returned, which cause the wait() to deblock the parent process.