

Practical Write Up

Elliott Rarden

March 16, 2016

1 Practical 1 - Pipes and Forks

1.1 Question 1

In the OSX implementation of `fork()`, processes do not call `clone`, so I cannot answer the expert question for this question. Instead, I will discuss why Mac OS does this. `fork()`'s implementation on OSX is tied directly to the system's underlying process manager, Grand Central Dispatch (GCD). Apple's engineers chose to use the `fork()` system call over `clone()` to work with GCD.

1.1.1 Strace output from Beej's code

```
$ sudo dtruss -acf -t process -t pipe ./beej
```

PID/THRD	RELATIVE	ELAPSD	CPU	SYSCALL(args)	= return
50591/0x8b0399:	6394	8	5	pipe(0x2, 0xC59F, 0x11)	= 3
50592/0x8b03a8:	109	0	0	fork()	= 0

CALL	COUNT
fork	1 // GCD at work
ioctl	1
munmap	1
pipe	1 // Pipe is created
proc_info	1
shared_region_check_np	1
bsdthread_register	2
dup	2 // Streams are copied
execve	2
fcntl	2
getpid	2
issetugid	2
pread	2
sysctl	2
csops	3
open	3 // FILE* is fun
thread_selfid	3
mmap	4 // Streams are set in memory
close	7
mprotect	8

1.2 Question 2

Any process can talk to any other if they set up the correct pipes. This is incredibly common in UNIX scripting. Pipes can even go to files (they are all FILE* in the end anyway)

```
$ ls -al > currentDirectoryListing.txt
```

1.2.1 My code for fork/pipe

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(void)
6 {
7     // index 0 = read
8     // index 1 = write
9     int pipe1[2];
10    int pipe2[2];
11    pipe(pipe1);
12    pipe(pipe2);
13
14    if (!fork()) // New process created here, it starts after the else
15    { // Process 0
16
17        // There is no input from pipe1, close it's input
18        close(pipe1[0]);
19
20        // Set stdout to pipe1
21        close(1);
22        dup(pipe1[1]);
23
24        // Make call
25        execlp("cat", "cat", "/etc/passwd", NULL);
26    }
27    else // the process created here spawns into two processes
28    {
29        if (!fork()) // Same as above, new process starts after the else
30        { // Process 1
31
32            // Set standard input from pipe1
33            close(0);
34            dup(pipe1[0]);
35
36            // set stdout to pipe2
37            close(1);
38            close(pipe1[1]);
39            dup(pipe2[1]);
40
41            // Make call
42            execlp("cut", "cut", "-f1", "-d:", NULL);
43        }
44        else
45        { // Process 2
46
47            // Set standard input from pipe2
48            close(0);
49            close(pipe1[0]);
50            dup(pipe2[0]);
51
52            // There is no output to pipe1 or pipe2
53            close(pipe1[1]);
54            close(pipe2[1]);
55
56            // Make call
57            execlp("sort", "sort", NULL);
58        }
59    }
60    // Done
61    return 0;
62 }
```

1.2.2 Strace output from my code

PID/THRD	RELATIVE	ELAPSD	CPU	SYSCALL(args)	= return
50708/0x8b2613:	5732	11	6	pipe(0x2, 0xC614, 0x11)	= 3
50708/0x8b2613:	5736	5	3	pipe(0x2, 0xC614, 0x11)	= 5
50709/0x8b2623:	74	0	0	fork()	= 0
50710/0x8b2624:	72	0:	0	fork()	= 0

CALL	COUNT
fstat64	1
ioctl	1
munmap	1
proc_info	1
shared_region_check_np	1
fcntl	2
fork	2 // Created 2 children
getpid	2
issetugid	2
pipe	2 // Creating pipes
pread	2
sysctl	2
bsdthread_register	3
// Arch BSD's Motto - "only after a few drinks"	
csops	3
execve	3
open	3
dup	4 // Opening FILE*
mmap	4
thread_selfid	4
mprotect	8
// GCD is really cool, it protects your children when you don't	
close	12 // You open 4, you close 8
stat64	45

2 Practical 2 - Concurrency

2.1 Question 1 - Concurrent processes

Multiple processes can have 100% processor usage if your machine has more than one physical core. Since the process queue can distribute its workload to multiple processors, they can each have 100% of a given core. You can use this knowledge to determine how many cores a processor has. Start by setting N equal to 2. If both processes reach 100%, then increase N to 4. Again, if both processes reach 100%, increase N to 8. Repeat until the processes have a value less than 100%. The largest number where the processors do not reach 100% is the number of cores.

2.1.1 Table of processor usage for Question 1

N (Number of Processes)	Approx Processor Usage (% per process)
2	100%
4	100%
8	50%
16	25%

2.2 Question 2

2.2.1 Code for Question 2

```
1  #include <unistd.h>
2  #include <pthread.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <math.h>
6
7  #define NUMTHREADS 2 /* define the total number of Threads we want */
8
9  /* Set global variable */
10 float total=0;
11
12 /* compute function just does something. */
13 void *compute()
14 {
15     int i;
16     float oldtotal=0, result=0;
17
18     /* for a large number of times just square root and square
19     the arbitrary number 1000 */
20     for(i=2000000000;i!=0;i--)
21     {
22         result=sqrt(1000.0)*sqrt(1000.0);
23     }
24
25     /* Print the result ? should be no surprise */
26     printf("Result is %f\n", result);
27
28     /* We want to keep a running total in the global variable total */
29     oldtotal = total;
30     total = oldtotal + result;
31
32     /* Print running total so far. */
33     printf("Total is %f\n", total);
34
35     pthread_exit(NULL);
36 }
37
38 int main()
39 {
40     pthread_t threads[NUMTHREADS];
41     int i;
42     int retcodes[NUMTHREADS];
43     float result=0;
44
45     printf("\n"); /* bit of whitespace */
46
47     /* Create the proper number of threads */
48     for(i=0; i<NUMTHREADS; i++)
49     {
50         /* give a message about which thread we're creating */
51         printf("Creating Thread-%d\n", i);
52
53         /* Create the thread */
54         retcodes[i] = pthread_create(&threads[i], NULL, compute, NULL);
55
56         if (retcodes[i]) // If pthread_create gave us a value that isn't 0
57         {
58             printf("ERROR; _return_code_from_pthread_create() is %d\n", retcodes[i]);
59             exit(-1); // Shit hit the fan
60         }
61     }
62
63     pthread_exit(NULL);
64
65     /* nothing else to do so end main function (and program) */
66     return 0;
67 }
```

2.3 Question 3

The difference in behaviour is that the threaded program keeps a running total between all of the threads, whereas the one with processes does not. This is because `fork()` creates an exact copy of the entire program, including the data section which includes global variables. The threaded version does not copy these, and instead, each thread uses the primary programs memory pool.

2.4 Question 4

The critical section is where the variable is updated.

2.4.1 Code for Question 4

From above, replace line 30 with

```
1      /* Critical Section - add POSIX semaphore */
2      #if USE_SEMAPHORES != 0
3          sem_wait(&mutex);
4          total = oldtotal + result;
5          sem_post(&mutex);
6      #else
7          total = oldtotal + result;
8      #endif
```

Add the following to `main()` before the `for` loop

```
1      #if USE_SEMAPHORES != 0
2          // Initialise the semaphore
3          sem_init(&mutex, 0, 1);
4      #endif
```

And add the following after the `pthread_exit(NULL);` statement

```
1      #if USE_SEMAPHORES != 0
2          /* Once all threads are merged, we can destroy the semaphore */
3          sem_destroy(&mutex);
4      #endif
```

You will also need to declare the following globally, near the top of the file

```
1      // Partial compilation
2      #define USE_SEMAPHORES 1
3
4      #if USE_SEMAPHORES != 0
5          // Initialise the semaphore
6          sem_t mutex;
7      #endif
```

You can enable or disable the semaphore by setting `USE_SEMAPHORES` before compilation to either 1 (enable) or 0 (disable).

3 Practical 3 - Interprocess Communication

3.1 Question 1

3.1.1 kirk2.c

```
1  /*
2  ** kirk2.c — writes to a message queue
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <sys/ipc.h>
11 #include <sys/msg.h>
12
13 // Message Types
14 #define URGENT (1)
15 #define NORMAL (2)
16
17 struct my_msgbuf {
18     long mtype;
19     char mtext[200];
20 };
21
22 int main(void)
23 {
24     struct my_msgbuf buf;
25     int msqid;
26     key_t key;
27
28     if ((key = ftok("kirk.c", 'B')) == -1) {
29         perror("ftok");
30         exit(1);
31     }
32
33     if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
34         perror("msgget");
35         exit(1);
36     }
37
38     printf("Enter lines of text, ^D to quit:\n");
39
40     while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL) {
41         int len = strlen(buf.mtext);
42
43         /* ditch newline at end, if it exists */
44         if (buf.mtext[len-1] == '\n')
45             buf.mtext[len-1] = '\0';
46
47         /* Check and set message type */
48         if (len > 6 && (strncmp("URGENT", buf.mtext, 6) == 0) )
49             buf.mtype = URGENT;
50         else
51             buf.mtype = NORMAL;
52
53         if (msgsnd(msqid, &buf, len+1, 0) == -1) /* +1 for '\0' */
54             perror("msgsnd");
55     }
56
57     if (msgctl(msqid, IPC_RMID, NULL) == -1) {
58         perror("msgctl");
59         exit(1);
60     }
61
62     return 0;
63 }
```

3.1.2 starfleet.c

```
1  /*
2  ** starfleet.c — reads from a message queue
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <errno.h>
8  #include <sys/types.h>
9  #include <sys/ipc.h>
10 #include <sys/msg.h>
11
12 // Message Types
13 #define URGENT (1)
14 #define NORMAL (2)
15
16 struct my_msgbuf {
17     long mtype;
18     char mtext[200];
19 };
20
21 int main(void)
22 {
23     struct my_msgbuf buf;
24     int msqid;
25     key_t key;
26
27     if ((key = ftok("kirk.c", 'B')) == -1) { /* same key as kirk.c */
28         perror("ftok");
29         exit(1);
30     }
31
32     if ((msqid = msgget(key, 0644)) == -1) { /* connect to the queue */
33         perror("msgget");
34         exit(1);
35     }
36
37     printf("starfleet: ready to receive messages, captain.\n");
38
39     for(;;) { /* Spock never quits! */
40         if (msgrcv(msqid, &buf, sizeof buf.mtext, URGENT, 0) == -1) {
41             perror("msgrcv");
42             exit(1);
43         }
44         printf("starfleet: \"%s\"\n", buf.mtext);
45     }
46
47     return 0;
48 }
```


3.2 Question 2

3.2.1 kirk3.c

```
1  /*
2  ** kirk3.c -- teleportation?
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <sys/ipc.h>
11 #include <sys/msg.h>
12 #include <signal.h>
13
14 // Message Types
15 #define URGENT (1)
16 #define NORMAL (2)
17
18 struct my_msgbuf {
19     long mtype;
20     char mtext[200];
21 };
22
23 volatile sig_atomic_t got_usr1;
24
25 void sigusr1_handler(int sig)
26 {
27     got_usr1 = 1;
28 }
29
30 int main(void)
31 {
32     struct sigaction sa;
33     got_usr1 = 0;
34
35     sa.sa_handler = sigusr1_handler;
36     sa.sa_flags = 0;
37     sigemptyset(&sa.sa_mask);
38     if (sigaction(SIGUSR1, &sa, NULL) == -1) {
39         perror("sigaction");
40         exit(1);
41     }
42
43     struct my_msgbuf buf;
44     int msqid;
45     key_t key;
46
47     if ((key = ftok("kirk.c", 'B')) == -1) {
48         perror("ftok");
49         exit(1);
50     }
51
52     if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
53         perror("msgget");
54         exit(1);
55     }
56
57     printf("Enter_lines_of_text, _^D_to_quit:\n");
58
59     while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL && !got_usr1) {
60         int len = strlen(buf.mtext);
61
62         /* ditch newline at end, if it exists */
63         if (buf.mtext[len-1] == '\n')
64             buf.mtext[len-1] = '\0';
65
66         /* Check and set message type */
67         if (len > 6 && (strncmp("URGENT", buf.mtext, 6) == 0))
68             buf.mtype = URGENT;
69         else
70             buf.mtype = NORMAL;
71
72         if (msgsnd(msqid, &buf, len+1, 0) == -1) /* +1 for '\0' */
73             perror("msgsnd");
74     }
75
76     if (msgctl(msqid, IPC_RMID, NULL) == -1) {
77         perror("msgctl");
78         exit(1);
79     }
80
81     return 0;
82 }
```

```

Terminal -- -bash -- 101x27
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
msgget: No such file or directory
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
spock: ready to receive messages, captain.
spock: "Hello Spock"
spock: "I'm back"
msgrcv: Identifier removed
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
spock: ready to receive messages, captain.
spock: "Hello"
spock: "This is my way to send messages to spock"
spock: "See"
spock: "This is sending a message"
spock: "I can see system calls as they are sent"
spock can't send messages
spock: "Yay"
msgrcv: Identifier removed
Elliott@HackAir:~/dev/OS/practices/PS$ spock can't send messages
Elliott@HackAir:~/dev/OS/practices/PS$
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
spock: ready to receive messages, captain.
spock: "I have all three things set up"
spock: "Spock hears my messages"
spock: "Dtruss records system calls from ./kirk3"
spock: "And I can kill hte process using kill -USR1 $(pidof kirk3)"
msgrcv: Identifier removed
Elliott@HackAir:~/dev/OS/practices/PS$

Terminal -- -bash -- 101x27
kirk3.c:31:21: error: use of undeclared identifier 'buf'
    if (msgsnd(msgid, &buf, len, 0) == -1)
                        ^
5 errors generated.
Elliott@HackAir:~/dev/OS/practices/PS$ gcc -o kirk3 kirk3.c
Elliott@HackAir:~/dev/OS/practices/PS$ ./kirk3
Enter lines of text, ^D to quit:
Hello Spock
URGENT fuck
I'm back
Elliott@HackAir:~/dev/OS/practices/PS$ ./kirk3
Enter lines of text, ^D to quit:
Hello
This is my way to send messages to spock
URGENT messages don't go to spock
See
This is sending a message
I can see system calls as they are sent
Yay
Elliott@HackAir:~/dev/OS/practices/PS$ ./kirk3
Enter lines of text, ^D to quit:
I have all three things set up
Spock hears my messages
Dtruss records system calls from ./kirk3
And I can kill hte process using kill -USR1 $(pidof kirk3)
Elliott@HackAir:~/dev/OS/practices/PS$

Terminal -- -bash -- 101x24
spock: ready to receive messages, captain.
spock: "URGENT hello"
spock: "hello"
spock: "URGENT hello"
spock: "hello"
msgrcv: Identifier removed
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
msgget: No such file or directory
Elliott@HackAir:~/dev/OS/practices/PS$ gcc -o spock2 spock2.c
Elliott@HackAir:~/dev/OS/practices/PS$ ./spock2
spock: ready to receive messages, captain.
spock: "hello"
msgrcv: Identifier removed
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 kirk3
-bash: kill: kirk3: arguments must be process or job IDs
Elliott@HackAir:~/dev/OS/practices/PS$ pidof kirk3
50098
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 50098
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 50098
-bash: kill: (50098) - No such process
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 $(pidof kirk3)
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 $(pidof kirk3)
Elliott@HackAir:~/dev/OS/practices/PS$ kill -USR1 $(pidof kirk3)
Elliott@HackAir:~/dev/OS/practices/PS$

Terminal -- -bash -- 101x24
= 24 0
50287/0x8aca3b: 2731 100 11 msgsnd(0xF0000, 0x7FFF59C19468, 0x19) = 0 0
50287/0x8aca3b: 2766 8937307 25 read_nocancel(0x0, "Dtruss records system calls from ./kirk3\n0", 0x1000) = 41 0
50287/0x8aca3b: 2786 100 13 msgsnd(0xF0000, 0x7FFF59C19468, 0x2A) = 0 0
50287/0x8aca3b: 2822 16651726 24 read_nocancel(0x0, "And I can kill hte process using kill -USR1 $(pidof kirk3)\n0", 0x1000) = 59 0
50287/0x8aca3b: 2848 119 19 msgsnd(0xF0000, 0x7FFF59C19468, 0x3C) = 0 0
50287/0x8aca3b: 2878 2499800 26 read_nocancel(0x0, "\0", 0x1000) = -1 Err#4
50287/0x8aca3b: 2887 80 3 sigreturn(0x7FFF59C192E0, 0x1E, 0x1000) = 0
Err#2
50287/0x8aca3b: 2909 67 7 msgctl(0xF0000, 0x0, 0x0) = 0 0
dtrace: error on enabled probe ID 2199 (ID 895: syscall::thread_selfid:return): invalid user access 1
n action #5 at DIF offset 0
CALL COUNT
exit 1
msgctl 1
sigreturn 1
msgsnd 4
read_nocancel 4
Elliott@HackAir:~/dev/OS/practices/PS$

```

Figure 1: Windows clockwise starting in top left: spock.c, kirk3.c, dtruss (OSX strace), BASH (to give kill command).